Desarrollo de una app para la generación preventiva de alertas sanitarias

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Realizado por:

Diego Díaz Bailón

Manuel Martínez Sánchez

Pablo Panero

Director:

Antonio Sarasa Cabezuelo

Departamento de sistemas informáticos y computación - lenguajes y sistemas informáticos y ciencias de la computación e inteligencia artificial

Facultad de Informática

Universidad Complutense de Madrid

Madrid, Junio 2016

Autorización de difusión

lunes, 13 de junio de 2016

Los abajo firmantes, alumnos y tutor del Trabajo Fin de Grado (TFG) en el Grado en Ingeniería Informática de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado (TF): "Desarrollo de un sistema para la generación preventiva de alertas sanitarias", realizado durante el curso académico 2015-2016 bajo la dirección de Antonio Sarasa Cabezuelo en el Departamento de sistemas informáticos y computación - lenguajes y sistemas informáticos y ciencias de la computación e inteligencia artificial. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Diego Díaz Bailón Manuel Martínez Sánchez Pablo Panero

Antonio Sarasa Cabezuelo

Dedicatoria

"El ignorante afirma, el sabio duda y reflexiona "

Después de muchos meses de esfuerzo y trabajo, parece que el proyecto llega a su fin y no queremos terminarlo sin antes mencionar a todas aquellas personas que nos han motivado, ayudado, inspirado o simplemente escuchado en este ambicioso trabajo, lleno de desafíos y retos.

En primer lugar, queremos dedicar nuestro proyecto a nuestras familias, que seguramente se sentirán muy orgullosas de aparecer en estas líneas. En segundo lugar, a nuestro profesor y director Antonio, a nuestros amigos de la carrera, conscientes de la importancia de esta memoria y a nuestras novias, que nos han acompañado y aguantado en todo este tiempo.

A todos ellos gracias por estar ahí, por haber aparecido en nuestras vidas y por valorar tan positivamente nuestro trabajo realizado.

Agradecimientos

"Cada cosa tiene su belleza, pero no todos pueden verla"

Queremos dar las gracias, en primer lugar, a nuestro tutor y coordinador del proyecto, Antonio Sarasa Cabezuelo, por haberlo aceptado, confiar en nosotros y ayudarnos a sacarlo adelante. Su interés se ha visto reflejado en nuestro trabajo y en sus positivas iniciativas: la propuesta de publicación del proyecto en una revista científica y la presentación del mismo en un congreso internacional.

Queremos agradecer de una manera especial el esfuerzo, interés y dedicación del doctor Francisco López Medrano, médico adjunto del Hospital Universitario 12 de Octubre, el cual nos abrió la puerta de su despacho y no dudó en escuchar, opinar y asesorar en nuestro trabajo fin de grado. Queremos darle las gracias por sus opiniones, sus consejos y sus puntualizaciones, que han permitido que el proyecto adquiera una orientación más realista, práctica y útil en el ámbito de la medicina.

Queremos agradecer también la colaboración de Almudena Lagares Velasco, estudiante de medicina porque, sin su ayuda, habría sido imposible contactar con un profesional de la medicina y todos los detalles técnicos médicos se nos habrían escapado.

Por último, y no por ello menos importante, nuestra página web y nuestra API han podido darse a conocer al mundo instalándose en un servidor de la Facultad de Informática. De esta manera, todo nuestro sistema ha podido ser testeado en muchos dispositivos por gran cantidad de personas. Y el que ha llevado a cabo todo esto ha sido Joaquín Gayoso Cabada. Por ello, agradecerle de todo corazón el esfuerzo y la dedicación por pelearse con todas las tecnologías usadas y facilitarnos todo lo que le hemos pedido.

A todos ellos, nuestros más sinceros agradecimientos.

Prólogo

"Aprender sin pensar es inútil. Pensar sin aprender, peligroso"

Todo empezó un mes de junio cuando tres amigos, compañeros de carrera, decidieron emprender el que quizá fuera su último trabajo en la universidad. Esos tres amigos éramos nosotros, y ese gran proyecto era el trabajo fin de grado.

Duro ha sido el entrenamiento a lo largo de estos cuatro años, agotador el esfuerzo realizado y muy gratificante la conexión y la complicidad conseguida entre nosotros. Fruto de todo esto, surgió la idea de un trabajo innovador, ambicioso, en el que aprendiéramos nuevas tecnologías y del que sacáramos partido para nuestro futuro profesional. Empezó la lluvia de ideas y pronto el tema se declinó por la medicina... palabras como ébola, enfermedades, contagios, localización, análisis, etc, se repetían con frecuencia y de todo ello se forjó nuestro proyecto.

Este trabajo de fin de grado trata de abarcar el mundo del Big Data intentando analizar datos de forma masiva, con el fin de obtener aproximaciones de enfermedades y otros índices de interés, aprender Android desarrollando una app y dominar todo lo relacionado con la geolocalización gracias al GPS del móvil. Y todo ello implementarlo mediante tecnologías y lenguajes muy variados, siendo capaces de unir todos los módulos, conectarlos y hacer que trabajen en conjunto.

Sin saber si finalmente este proyecto verá la luz en el mercado en un futuro, si estará instalado en algún hospital del país y si miles de usuarios recibirán notificaciones periódicas en sus móviles, hemos luchado por elaborarlo y desarrollarlo de la mejor forma posible teniendo siempre presente la intención de ayudar a la medicina, a la detección de enfermedades y quien sabe, la posibilidad de llegar a salvar vidas humanas.

Resumen

Desde hace unos años, parece que la informática ha ido invadiendo numerosos campos de la ciencia, se ha ido consolidando como parte fundamental en el desarrollo y la tecnología y ya se puede ver como motor de la economía, las comunicaciones, el comercio e incluso, la medicina. Este trabajo se centra en la aplicación de la informática en el área de las enfermedades infecciosas, intentando contribuir con una nueva idea que revolucione el mundo de las mismas dentro de la medicina, en una situación en la que el término "Smart City" cada vez es más importante.

Se trata de un sistema formado por una app para Android y una web controlada y gestionada por un médico, la cual permite identificar posibles usuarios contagiados, localizar focos de contagios y gestionar de una manera óptima, el estado y los avisos de cada usuario.

En esta memoria se describe el trabajo realizado, la arquitectura del sistema, las tecnologías utilizadas, el diseño de la aplicación móvil y de la página web, su implementación, las pruebas realizadas, las conclusiones a las que se ha llegado y las posibles mejoras que se podrían incluir en el proyecto en un futuro.

Palabras clave

Alertas epidemiológicas, Control de enfermedades, Geolocalización, Big Data, API RESTful, Aplicación móvil, Aplicación web

Abstract

Since a few years, computer science has been introduced in many scientific fields, and has been consolidated as a fundamental part in their development and their technology. We can now understand it as a motor of the economy, communications, commerce and even medicine. Here is where we stopped and decide to contribute with a new and revolutionary idea in the infectious diseases' world, inside the medicine scope. We can here talk about "Smart Cities" and our project was thought as part of one of those.

The system consists of an application for mobile devices and a web application controlled and managed by medical experts. It gives them the possibility to identify users that problably have gotten infected, locate the focus of an outbreak and manage in an optimal way the state and notifications of each users.

In this document we try to explain all the work done, identify various parts where we talked about system architectur, used technologies, mobile and web design, its implementation, the tests that took place, the conclusions of out work and the possible improvements that could take the project further.

Keywords

Epidemological alerts, Disease Control, Geolocation, Big Data, API RESTful, Mobile application, Web application

Índice

Índice de figuras

Capítulo 1. Introducción

Con el fin de llevar a cabo el Trabajo Fin de Grado correspondiente al Grado en Ingeniería Informática, se ha pensado una idea que surgió gracias al interés en dos temas fundamentales. En primer lugar, el deseo de llevar a cabo algún proyecto relacionado con el mundo Big Data, Cloud y el análisis de datos masivos; y en segundo lugar, el aprendizaje y manejo de bases de datos no relacionales, con el fin de aprender y familiarizarse con este tipo de tecnología, cada vez con más fuerza y empuje.

Con todo esto presente, y después de que llegara a España una enfermedad contagiosa procedente de África llamada ébola, surgió la idea de crear un sistema capaz de identificar a un usuario contagiado, facilitar los individuos que hayan podido ser contagiados por dicha persona y filtrar esa búsqueda gracias a diversos parámetros como la enfermedad, la distancia o el período de exposición. Con el propósito de ayudar al campo de la sanidad, de facilitar a los médicos el trabajo del aislamiento, de buscar todas esas enfermedades y a todas esas personas contagiadas, se ha planteado llevar a cabo todas estas acciones para que se puedan realizar en el menor tiempo posible.

1.1 Motivación

En los últimos años se han producido a lo largo del mundo diferentes epidemias víricas que han dejado gran cantidad de muertos y afectados. Actualmente el virus del Zica se expande por Sudamerica, pero no hace mucho tiempo España se vio afectada por el Ébola. Frente a esta situación, se pensó en desarrollar un sistema informático capaz de identificar a un usuario contagiado, facilitar los individuos que hayan podido ser contagiados por dicha persona y filtrar esa búsqueda gracias a diversos parámetros como la enfermedad, la distancia o el período de exposición. Este sistema facilitaría a los médicos el trabajo del aislamiento, la búsqueda de los focos infecciosos y las personas contagiadas, con el objetivo de llevar a cabo acciones preventivas en el menor tiempo posible.

1.2 Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación móvil y una página web que ayude a los ciudadanos y a los médicos en el diagnóstico y la prevención de enfermedades infecciosas.

La página web debería ser fácil de utilizar para el médico de forma que pueda aprovechar todas sus funcionalidades. Así mismo, la app debería mostrar un aspecto moderno y atractivo, de forma que motive a los usuarios a utilizarla.

Los objetivos específicos planteados en el proyecto son los siguientes:

- Elaborar una aplicación móvil atractiva que capte la atención de los usuarios, ofreciendo servicios tales como actualizar las noticias y alertas, el estado actual del usuario y los brotes activos en un momento dado.
- Conseguir extraer información de geolocalización de un móvil, de forma que se pueda enviar la localización de cada usuario, guardarla en una base de datos y analizarla posteriormente para descubrir posibles contagios.
- Elaborar una web orientada a los médicos que sea sencilla y proporcione las funcionalidades necesarias para la identificación de un posible foco o la localización de las personas que han podido ser contagiadas por un individuo identificado.
- Diseñar y elaborar dos bases de datos, una relacional para almacenar la información referente a todos los usuarios, médicos, enfermedades y contagios y otra no relacional, donde se almacenarán las localizaciones de cada usuario de la aplicación móvil.
- Implementar una API RESTful que conecte y abstraiga todas las consultas a las bases de datos y las peticiones de la web y la aplicación.
- Diseñar e implementar dos algoritmos, uno que permita la identificación geográfica del foco de una enfermedad y otro que permita la localización de todas las posibles personas que hayan podido ser contagiadas por un usuario infectado mediante las localizaciones almacenadas.

1.3 Estado del arte

Ante la búsqueda de sistemas y aplicaciones similares del ámbito de la medicina no se ha encontrado mucha información. Parece que el proyecto no tiene competidores directos [30] [31]. Quizá lo más parecido sea una aplicación que ha sacado Sanitas [33], empresa médica privada donde se puede consultar los datos del paciente y tener una especie de agenda médica. Hace pocos meses también se ha desarrollado en Valencia, en un hospital, una aplicación en tablets donde las enfermeras pueden llevar un histórico y un registro de cada habitación y tener información de los pacientes en tiempo real [32] [34].

1.4 Estructura de la memoria

A continuación se explica brevemente la estructura de la memoria:

- **Capítulo 1: Introducción**
  En este capítulo se realiza una exposición breve del proyecto mediante un resumen y una serie de apartados y, además, se detalla la estructura que seguirá este documento.

- **Capítulo 2: Especificación de la aplicación**
  En este capítulo se detallan las funcionalidades del sistema, tanto de la aplicación móvil y de la web, como del recolector de datos y de las APIs. Además se incluyen los diagramas de actividades de cada una.

- **Capítulo 3: Tecnologías empleadas**
  En este capítulo se listan las tecnologías empleadas y se describen sus principales características.

- **Capítulo 4: Arquitectura de la aplicación**
  En este capítulo se detallan la organización de los distintos módulos que conforman el sistema.

- **Capítulo 5: Modelo de datos**
  En este capítulo se profundiza en la estructura y organización que siguen las bases de datos, tanto las relacionales como las no relacionales.

- **Capítulo 6: Diseño**
  En este capítulo se explica todo aquello relacionado con el diseño del sistema: casos de uso, patrones de diseño utilizados, etc. Todo ello se expone indicando las herramientas usadas y remarcando las diferencias de diseño de la web frente a la aplicación móvil.

- **Capítulo 7: Implementación**
  En este capítulo se profundiza en cómo se ha llevado a cabo en el proyecto. Se trata del apartado más técnico, con un lenguaje muy específico en el que se describen todas las tecnologías usadas, se detallan los algoritmos usados, las conexiones entre los diferentes módulos y otros detalles acerca de los diferentes lenguajes, librerías o plugins utilizados.

- **Capítulo 8: Conclusiones y Trabajo futuro**
  Se presentan las conclusiones acerca del trabajo obtenidas después de la realización de diversas pruebas y testing comprobando la funcionalidad del producto; se resume la experiencia de los creadores y desarrolladores del trabajo y se detalla el trabajo individual realizado por cada uno de los integrantes del grupo. Por último, se enumeran las posibles mejoras que se podrían llevar a cabo con el fin de aumentar su competitividad en el mercado y mejorar su experiencia de uso.

- **Bibliografía**
  Como último apartado, se enumeran todas las referencias tomadas a lo largo de la elaboración del trabajo incluyendo libros, artículos o direcciones web de foros, tutoriales u otras páginas de interés.

- **Apéndice A: Manual de instalación**
  En este primer apéndice se explica paso a paso el proceso para instalar y configurar el sistema en su totalidad.

- **Apéndice B: Manual de usuario**
  En este apéndice se encuentra una serie de guías para el correcto uso de la aplicación por parte de los usuarios, y de la web por parte de los médicos o especialistas.

- **Anexo: Justificación del especialista médico**
  En este anexo se adjunta el documento que acredita la reunión con el especialista y su implicación en el proyecto.

Chapter 1. Introduction

In order to carry out the Bachelor Thesis for the Computer Scienci Engineering Degree, an idea came up thanks to the interest in two main topics. Firstly, our willing to carry out a project that reached the Big Data, Data Mining and Cloud Computing world. Secondly, our willing to learn how to deal with non-relational databases and similar technologies that are getting more and more important each day.

Having all that in mind, and after an important disease, Ebola, reached Spain coming from Africa, the idea of a system that could identify who has been infected in order and to whom it could have been spread being able to filter the search by several parameters that affects a particular disease such as exposure time or distance grew up in strength. It could help, or at least try to, the healthcare system, easing doctors' task in isolating, searching and trating of all posible infected people. The objective was to reduce the amount of time spent in those tasks to the mínimum possible.

1.1 Motivation

In recent years several outbreaks have been spreading over the world having a great impact in society in terms of dead and affected people. Nowadays the Zika virus is spreading all through south America, but not long ago Spain was facing an Ebola outbreak. In this sense, the idea to develop a system capable of identifying which user is infected, and to whom did those spread the disease, being able to filter the search by parameters like disease, distance or time of exposure came up. This system would ease medical specialist isolate, search and manage infected people, reducing the time spent in those tasks to the minimum.

1.2 Objectives

The main objective of the system, conformed by a mobile and a web application, is to help citeizens and doctors diagnose and prevent infectious diseases.

The general objective is to develop an easy to use web page through which the doctor can make use of all its functionalities. In addition, the development of and Andoird application that uses *Material Design*, in order to make it more attractive to users.

1.2.1 Specific Objectives:

- Develop a mobile application for the Android platform, using *Material Design* in order to make it more appealing to the users to they could engage more by being up-to-date on contagions, alerts and outbreaks.
- Make use of the GPS sensor in mobile devices. Being able to get each users' location, store it and use it to predict contagions.
- Develop a web application, using *Bootsrap,* for the medical experts. It has to be easy to use and give the necessary functionalities for focus location and user contagion identification.
- Design and implement several databases in order to store the iformation of users, doctors, diseases, contagions. They will be relational and non-relationl depending on the structured charateristic of the data.
- Implement a RESTful API that connects and abstracst the interaction with the databases and the data processing from the web and mobile application.
- Design and implement two algorithms, core of out project, they consists of the identification of posible foucs' places and users' contagions.

1.3 State of the art

After researching for similar applications or systems we did not find related information. It seems that our project that not has any diret competitors [30] [31]. Maybe an applications developed by Sanitas could be similar [33]. It allows direct contact between doctors and patients and stores the medical agenda. A similar system was also developed in a Valencian hospital though which nurses and doctors could track and have nearly real-time information about their patients [32] [34].

1.4 Memory Structure

- **Chapter 1: Introduction**
  In this chaper it is exposed a brief explanation of the project and the structure of this document.

- **Chapter 2: Application specification**
  In this capter the functionalities of the system are detailed, both the web and mobile application, the data collector and the two APIs. Activity diagrams are also included.

- **Chapter 3: Used technologies**
  In this chapter the chosen technologies are explained and briefly compared.

- **Chapter 4: Application architecture**
  In this chapter the different modules that conform the system are detailed.

- **Chapter 5: Data model**
  In this chapter there is an in depth explanation of the design and organization of both relational and non-relation used databases.

- **Chapter 6: Design**
  In this capter evety aspect related to the design is explained: use cases, design patterns, etc. All these is explained detailing the used tools and the diferences between web and mobile application.

- **Chapter 7: Implementation**
  In this chapter there is an in depth explanation of how the Project has been carried out. It is the most technical part, using a more specific language to describe the modules and their connections, the algorithms, programming languages, libraries and plugins.

- **Chapter 8: Conclusions and future work**
  Conclussions from the wrok done are exposed after having done the testing phase, in order to prove the Project functionality. The experience of the developers is summed up and each ones' tasks are detailed. At last the possible improvements that could differentiate this project from others are pointed out.

- **Bibliography**
  In this last chapter sources used to carried out this Project, such as books, articles, tutorials, fórums or sources of interest are ennumerated.

- **Apendix A: Intalation manual**
  This apendix contains the instructions to install and configure the system in its totality.

- **Apendix B: Users' manual**
  In this apendix there is a series of guides and steps to learn the correct use of the mobile application and the web application, for medical experts.

- **Apendix C: Confirmation of the medical expert**
  In this apendix an acreditation of the meeting and implication of a medical expert in the Project is attached.

Capítulo 2. Especificación de la aplicación

2.1 Funcionalidad Android

Como ya hemos mencionado, la función principal de la aplicación Android será alimentar la base de datos de Mongo con las localizaciones de los usuarios. En este sentido, puede decirse que esta tarea es "invisible" a ojos del usuario. La funcionalidad con la que realmente podrá interactuar el usuario se podría clasificar en tres acciones: ver su estado actual y el de la zona en la que se encuentre: recibir información de noticias como los nuevos brotes de enfermedades que aparezcan o la extinción de los mismos y, en tercer lugar, revisar su información personal (nombre, apellidos, DNI, email, peso, etc.).

Siendo ésta la funcionalidad más importante, la aplicación también permite al usuario realizar otras acciones que se describirán en detalle a continuación.

FUNCIONALIDAD 1: Envío de la localización

Es importante destacar en primer lugar que en la base de datos de Mongo no almacenamos únicamente la localización, sino también el instante de tiempo de la misma, es decir, vamos almacenando dónde estuvo el usuario y cuándo estuvo en ese punto.

Así pues, el envío de la localización se realiza cada 5 minutos o cada vez que se detecte un cambio de posición de 100 metros. De esta manera se evita saturar el envío de datos y mantener un control de los mismos y, sobre todo, se consigue ahorrar batería, un aspecto a tener en cuenta en los dispositivos móviles. Por ejemplo sería muy poco eficiente estar mandando la localización de manera continua mientras el usuario está durmiendo, ya que la localización sería la misma durante un largo periodo de tiempo (toda esta explicación será profundizada en detalle en el apartado Algoritmos del Capítulo 7).

FUNCIONALIDAD 2: Consulta del estado actual del usuario y de la zona

La primera de las tres principales funcionalidades con las que el usuario puede interactuar. Esta se encuentra en la primera pestaña de la aplicación (ver Figura 2.1.1), titulada 'Estado' y en ella se pueden ver dos tarjetas: la superior indica el estado actual del usuario, mientras que la inferior informa del estado de la zona en la que se encuentra en ese momento el usuario.

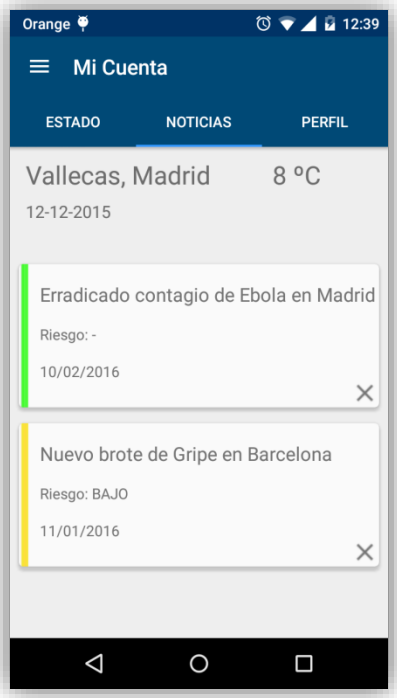
Como hemos dicho, la tarjeta superior le indica al usuario si su salud puede correr peligro porque el algoritmo así lo ha detectado o, si por el contrario, no lo corre. La inferior por su parte muestra si en la zona actual existe algún riesgo de contagio que se haya detectado por nuestro algoritmo o si la zona está libre de enfermedades.

Realizando el simple gesto de '*swipe*' (deslizar la pantalla hacia abajo) se actualizará la vista actual, de manera que se volverá a coger la posición actual del usuario para ver si en su nueva localización existe algún riesgo. Esto es útil tanto a pequeña escala en barrios o municipios, como si realizamos un viaje a otra comunidad. Podremos estar informados en todo momento si estamos expuestos a algún contagio.

1 - Figura 2.1.1 - Consultar estado

FUNCIONALIDAD 3: Consultar y eliminar noticias

Se encuentra en la segunda pestaña de la aplicación (ver Figura 2.1.2) y nos ofrece un listado de noticias interesantes, como la aparición de nuevos contagios o la extinción de los mismos. Se trata de noticias comunes para todos los usuarios, sin embargo, proporcionamos la posibilidad de eliminar una noticia. De esta forma, el usuario puede ir "limpiando" de la lista



2 - Figura 2.1.2 - Consultar noticias

aquellas noticias que ya haya revisado o que ya no le resulten de utilidad, y éstas dejaran de aparecerle. Esta acción se lleva a cabo presionando la X de la tarjeta de la noticia elegida.

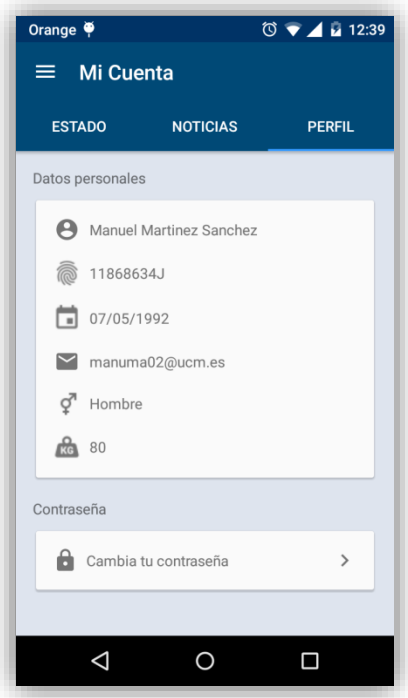
Además, en la parte superior de esta misma vista se encuentran datos de posible interés como lo son el municipio en el que te encuentras, la temperatura actual y la fecha.

Al igual que en la funcionalidad anterior, para actualizar toda la información explicada basta con realizar el gesto '*swipe*'.

FUNCIONALIDAD 4: Consultar perfil

Y en tercer lugar nos encontramos con la funcionalidad recogida en la tercera de las pestañas de la vista principal (ver Figura 2.1.3). Aquí podremos consultar los datos que solicitamos en el registro y que, más adelante, se podrán actualizar o modificar.

Actualmente no está implementada la actualización de estos datos, pero es algo que se recoge en el apartado de Trabajo futuro, por tanto, a día de hoy se trata de una funcionalidad meramente informativa.



3 - Figura 2.1.3 - Consultar perfil

FUNCIONALIDAD 5: Cambiar el idioma

Nos encontramos ante una funcionalidad pensada para aquellas personas extranjeras que lleguen a España y quieran usar y entender correctamente la aplicación. Actualmente se encuentra en inglés y en español, pero más adelante se podrán ir añadiendo más idiomas de manera simple gracias a la manera de implementar esta funcionalidad.

El cambio de idioma se realiza de manera rápida en los 'Ajustes' de la aplicación en el apartado de 'Idioma' (ver Figura 2.1.4). Inicialmente el idioma de la aplicación será el que este configurado por defecto en el dispositivo del usuario, pero en el momento en que éste lo desee cambiar se guardará en las preferencias de la aplicación. La explicación paso a paso se detallará en el Anexo II: Manual del usuario.

4 - Figura 2.1.4 - Cambiar idioma

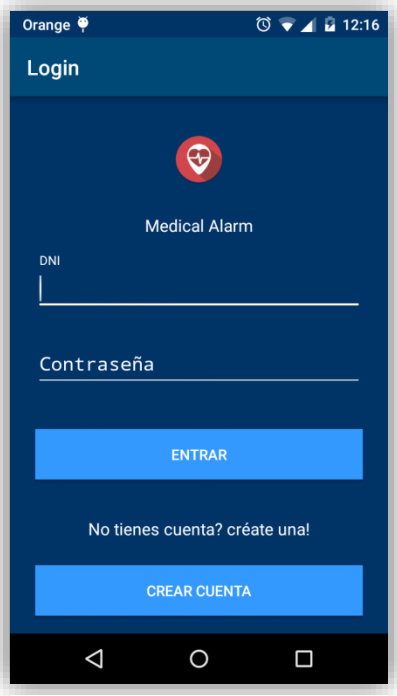
FUNCIONALIDAD 6: Login

Tanto la primera vez que ejecutemos la aplicación como cuando no tengamos guardados en las preferencias el usuario y la contraseña, nos aparecerá el *login*, mediante el cual iniciaremos la aplicación con nuestro usuario (ver Figura 2.1.5).

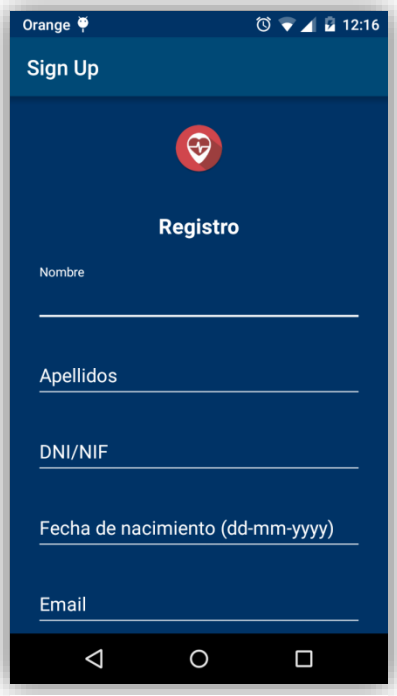
Se trata de un login simple que nos solicita nuestro DNI y contraseña para poder identificarnos. Además, en la parte inferior de la pantalla disponemos del botón para crearnos una cuenta en caso de no disponer una, se explica a continuación.

FUNCIONALIDAD 7: Registro

Como acabamos de mencionar, en caso de no poseer cuenta podemos crearnos una. En el registro se solicita el nombre y los apellidos, el email, el DNI, la fecha de nacimiento, el peso y la estatura y la contraseña (ver Figura 2.1.6). Todos los campos son necesarios y comprueban que los datos introducidos sean correctos.



6 - Figura 2.1.5 - Login



5 - Figura 2.1.6 - Registro

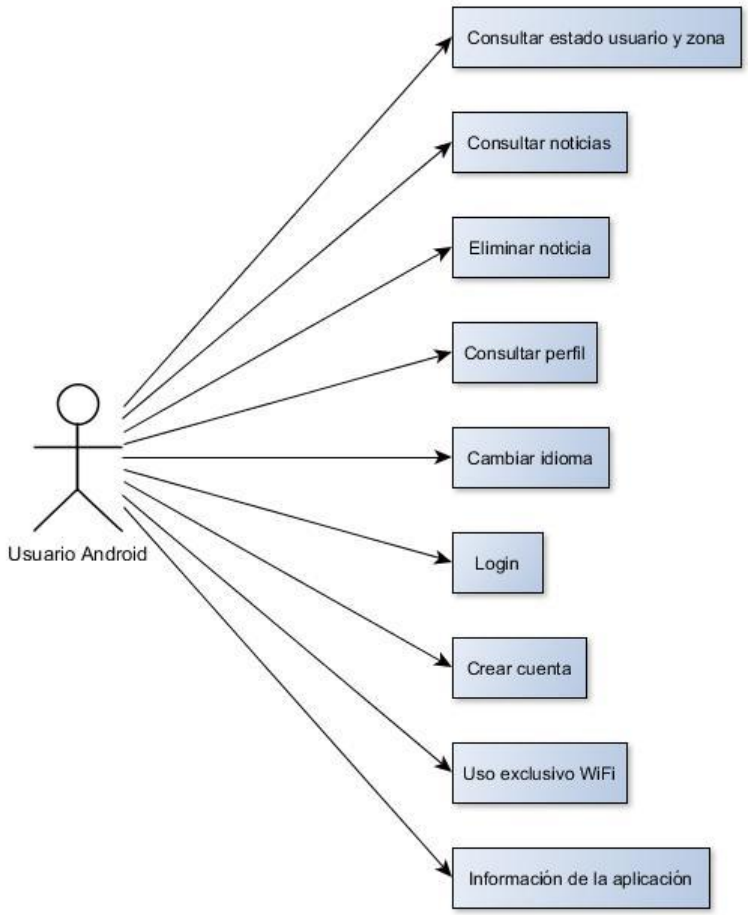
FUNCIONALIDAD 8: Activar/desactivar envío de datos de los sensores

Esta fue una funcionalidad que surgió a raíz de pensar que el usuario se quedase sin datos móviles, o simplemente que quisiera dejar de enviar los datos de sus localizaciones y de los sensores. Así pues, en el menú de Ajustes añadimos un interruptor o *'switch'* mediante el cual poder activar o desactivar el envío de datos de los sensores.

FUNCIONALIDAD 9: Obtener información sobre la aplicación

En esta funcionalidad se a engloban dos de las opciones que aparecen en el menú deslizante de la izquierda: 'Soporte' y 'Sobre nosotros'. En la primera se puede obtener información legal de la aplicación, términos de uso y se tiene como objetivo futuro implementar la sección de 'Preguntas frecuentes'. Por su parte, si pulsamos en 'Sobre nosotros' nos aparece una vista con una breve información relativa a los integrantes del desarrollo y de la aplicación en sí.

Así pues, a modo de resumen de las funcionalidades de la aplicación Android, se adjunta el siguiente diagrama de casos de uso[1] (ver Figura 2.1.7), que no solo resume, sino que también agrupa de un modo muy visual cuales han sido las funciones que el usuario podrá realizar con este sistema.

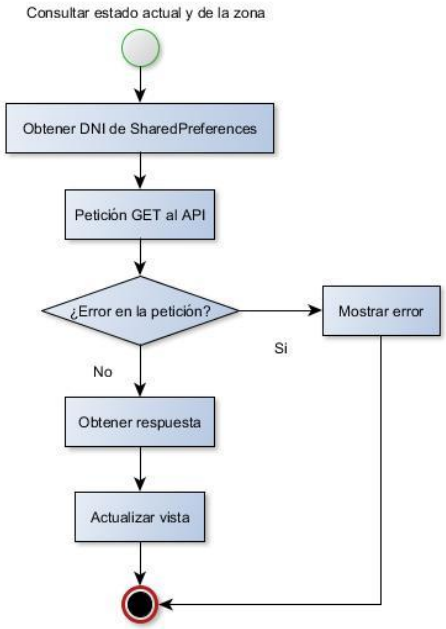


7 - Figura 2.1.7 - Casos de uso Android

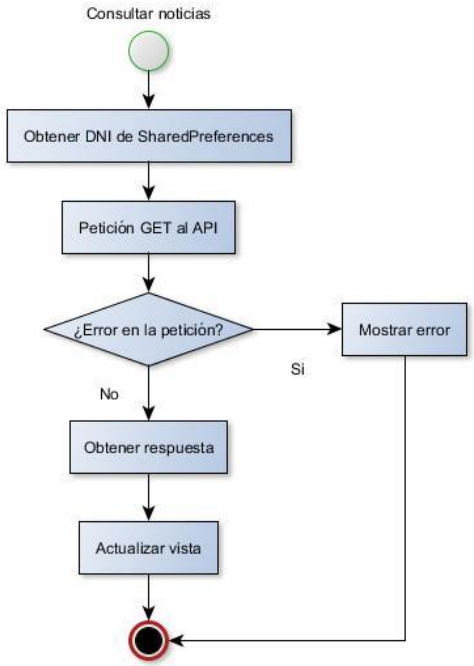
[1] representa la forma en como un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactuan

2.1.1 Diagramas de actividades

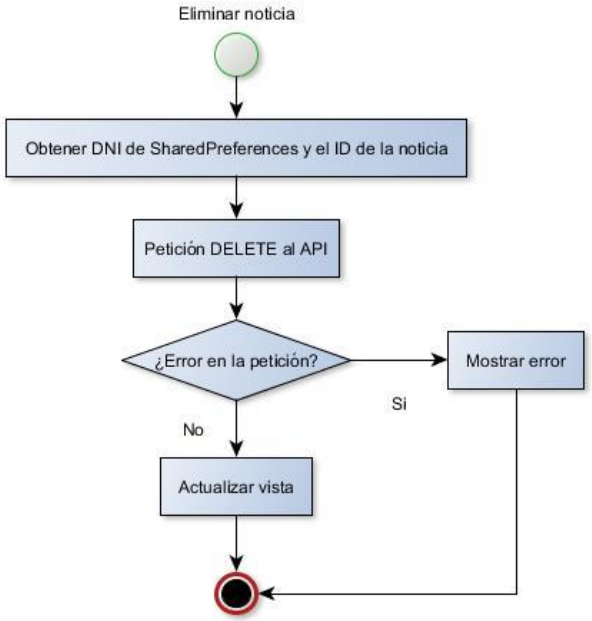
Consulta del estado actual del usuario y de la zona:



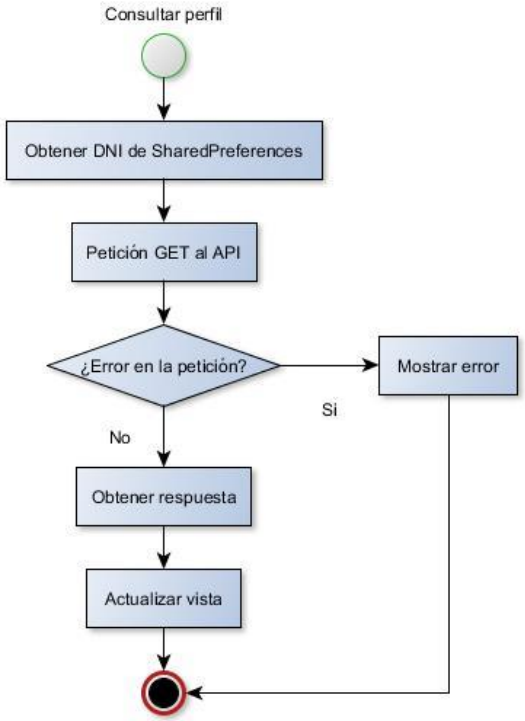
Consultar noticias:



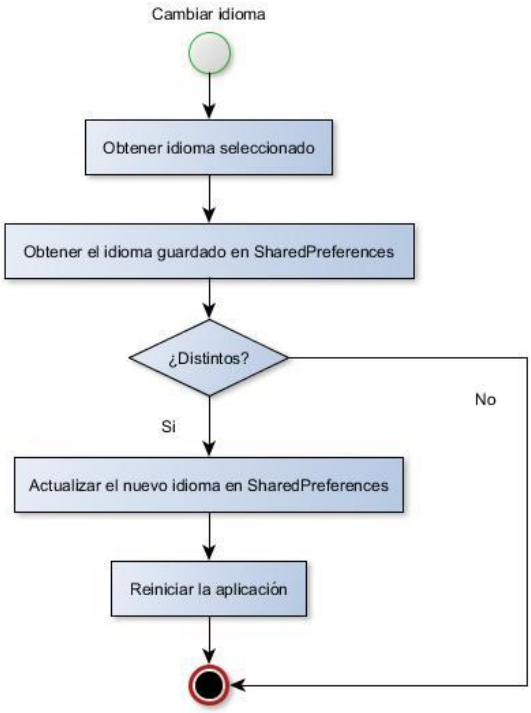
Eliminar noticias:



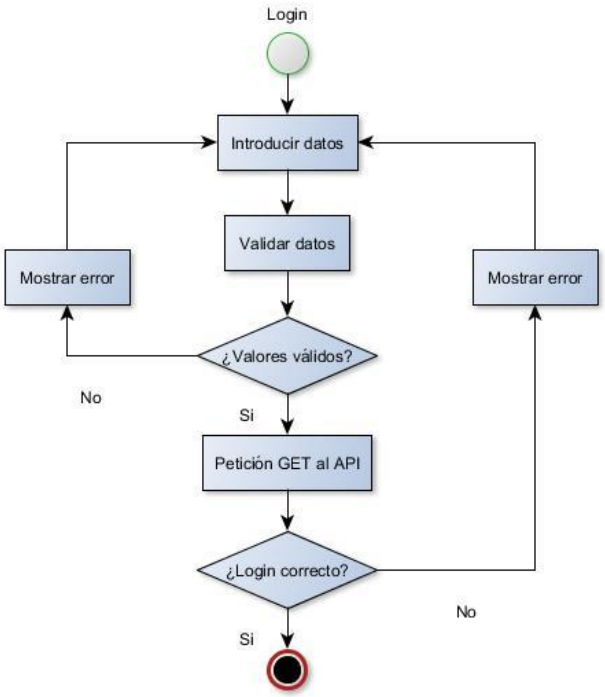
Consultar perfil:



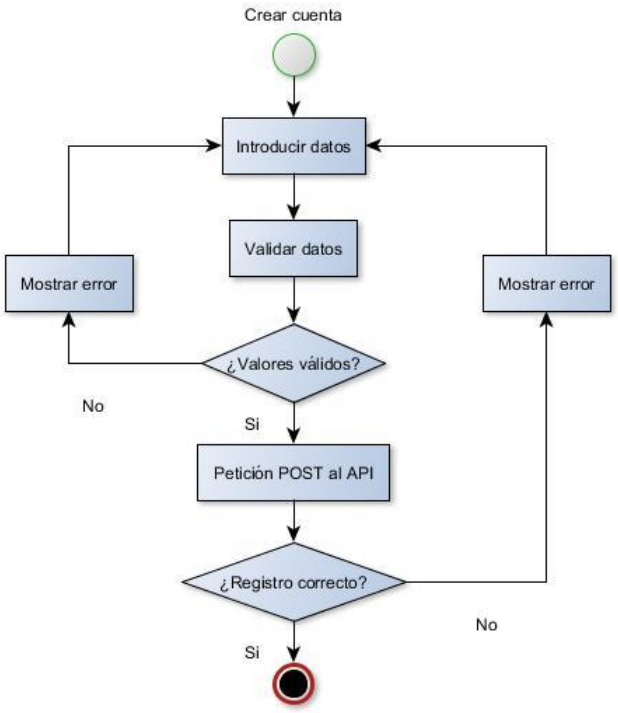
Cambiar el idioma:



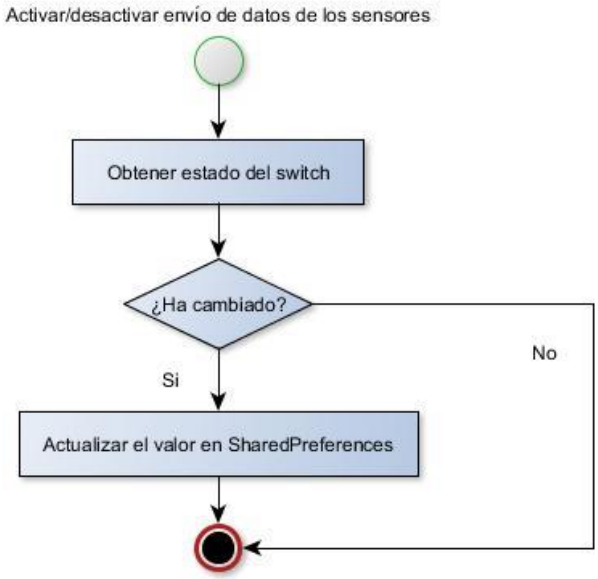
Login:



Registro:



Activar/desactivar envío de datos de los sensores:

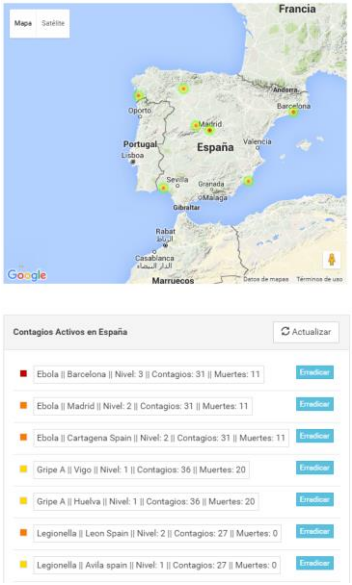


2.2 Funcionalidad página web

La página web es la herramienta con la que va a contar el médico día a día, la consultará todos los días y formará parte de su trabajo. Por esta razón, la página web fue diseñada para ser fácil de utilizar, intuitiva y cómoda y además llena de funcionalidades prácticas y útiles para agilizar y mejorar el trabajo de los médicos. Todas las acciones posibles tienen como objetivo el estudio de las distintas enfermedades, contagios y focos y usando los datos de los pacientes, mostrar toda esta información una vez computada de una manera clara y precisa. La página web se comunica con la API, la cual, consultando las bases de datos del sistema, le devuelve toda la información solicitada. A continuación se detalla cada una de las funcionalidades que la web ofrece:

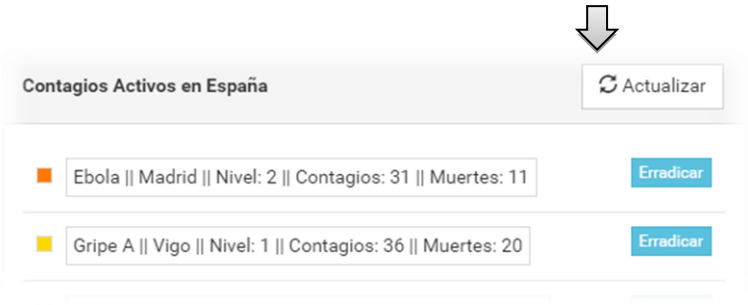
FUNCIONALIDAD 1: Ver contagios activos en España

Una de las funcionalidades más atractivas de la página web es la existencia de un mapa de calor (ver Figura 2.2.1) situado en la portada de la página donde se pueden visualizar los contagios activos en España en ese momento. Se trata de una información en tiempo real y viene acompañado de una leyenda en su parte inferior. En ella se puede ver en la parte izquierda el color del contagio, indicando la gravedad del mismo (rojo, naranja, amarillo), el nombre de la enfermedad, el nivel de alerta, el número de personas contagiadas y las muertes ocasionadas hasta el momento. Si se precisa más detalle, el mapa permite hacer zoom y trae varias opciones pudiendo alternar entre diferentes vistas del mapa: político o físico.



8 - Figura 2.2.1 - Ver contagios activos

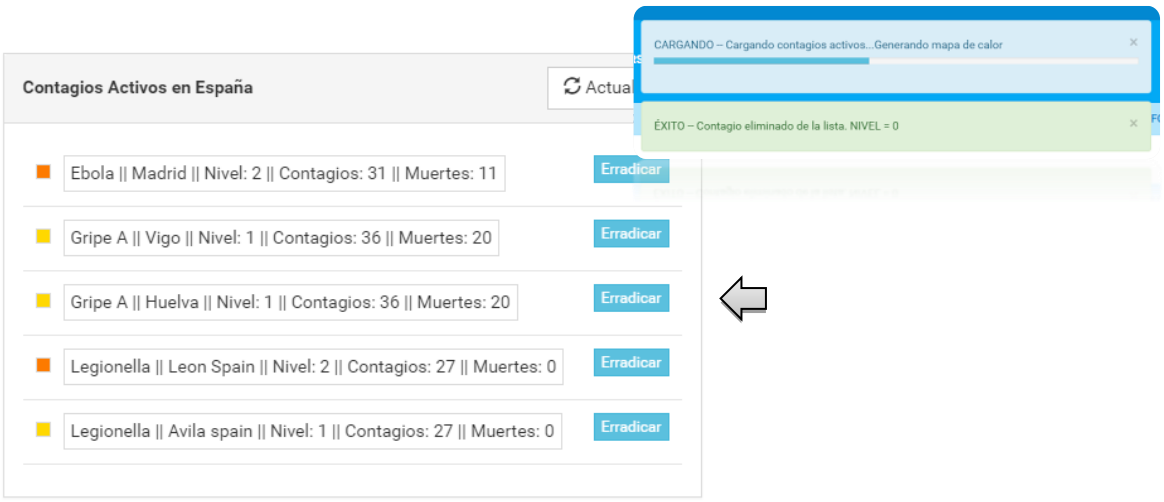
FUNCIONALIDAD 2: Actualizar los contagios activos en España



9 - Figura 2.2.2 - Actualizar contagios

Mediante el botón de Actualizar situado en la leyenda inferior del mapa de calor, dentro de la pestaña Inicio (ver Figura 2.2.2), se refresca la lista de contagios activos por si alguno de ellos ha sido modificado durante el período de observación.

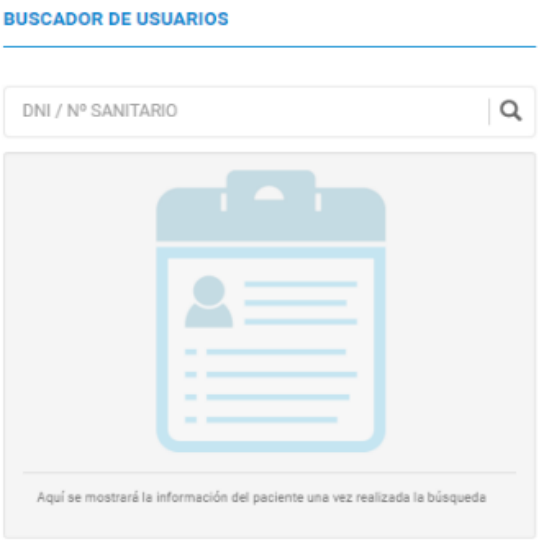
FUNCIONALIDAD 3: Erradicar un contagio activo en España



10 - Figura 2.2.3 - Erradicar contagio

En el mismo panel que la funcionalidad anterior, dentro del cuadro de contagios activos, para cada enfermedad activa hay un botón Erradicar (ver Figura 2.2.3) en el que, al pulsarlo, el nivel de alerta pasa a 0 y desaparece de la lista y del mapa de calor, mostrando un mensaje de alerta informando del cambio reciente.

FUNCIONALIDAD 4: Buscar un usuario en el sistema



12 - Figura 2.2.4 – Buscar usuario

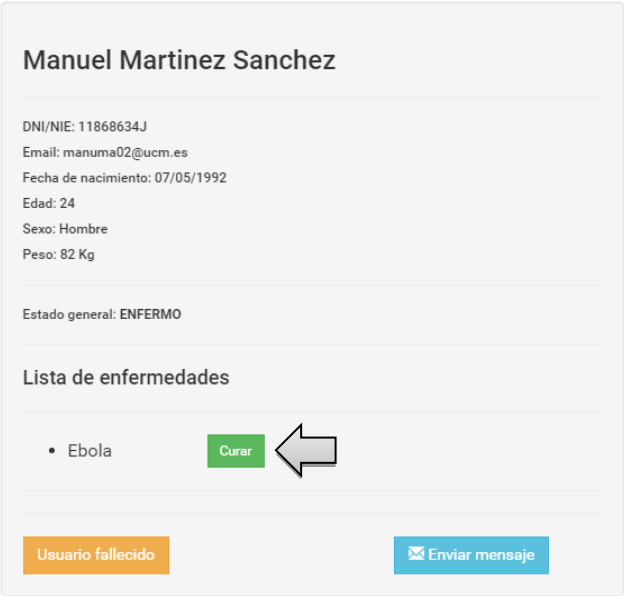
11 - Figura 2.2.5 – Cambiar estado usuario

En la parte superior derecha de la página de la pestaña de Inicio se encuentra un buscador, el cual permite buscar a un usuario registrado dentro del sistema. La búsqueda tiene lugar a través del DNI del paciente. En caso de no existir o de tratarse de un DNI incorrecto, se muestra un mensaje de advertencia y, en caso de ser correcto, se muestra la información detallada del paciente, como se puede apreciar en la imagen (ver Figura 2.2.4).

FUNCIONALIDAD 5: Cambiar estado de usuario a fallecido

Partiendo de la funcionalidad anterior, la referente a la búsqueda de un paciente, una vez encontrado se muestra un botón de Usuario fallecido (ver Figura 2.2.5), el cual al pulsarlo actualiza el estado del paciente a fallecido para que ese dato pueda ser tratado en las estadísticas correspondientes.

FUNCIONALIDAD_6: Eliminar enfermedad de un paciente

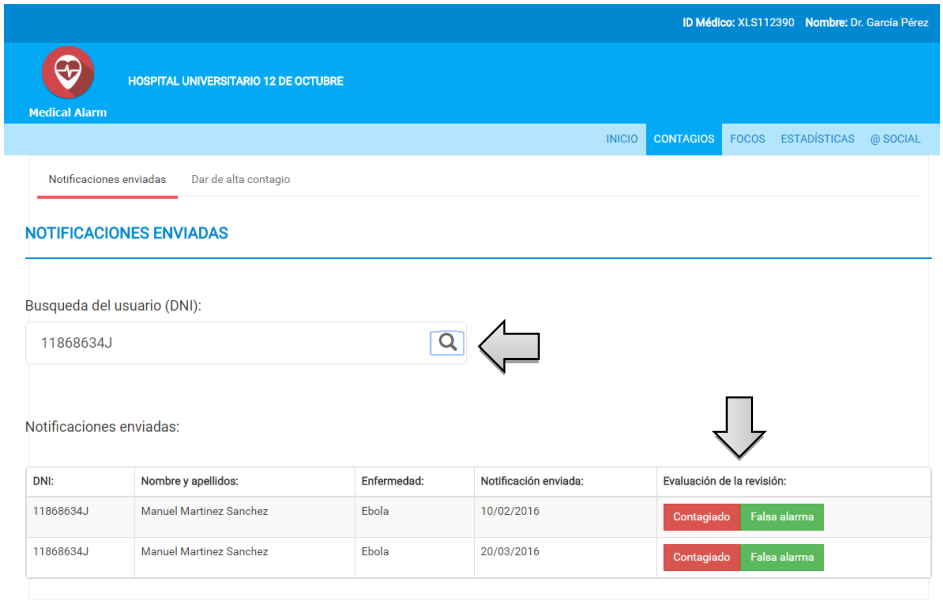


13 - Figura 2.2.6 – Eliminar enfermedad

Si el paciente buscado está enfermo, se muestran todas las enfermedades de las que está contagiado con un botón Curar (ver Figura 2.2.6) donde el médico puede pulsar para eliminar esa enfermedad del paciente en caso de haberse curado. Si desaparecen todas las enfermedades del paciente, el estado del mismo se actualiza de enfermo a curado.

FUNCIONALIDAD 7: Mostrar notificaciones enviadas a un paciente

En la pestaña de Contagios, nos encontramos con un buscador de usuarios por DNI, el cual permite mostrar todas las notificaciones enviadas al usuario buscado. Se muestra una tabla con el DNI, el nombre, la enfermedad por la que se ha notificado y la fecha en la que fue enviada. Mediante dos botones se ofrece al médico la posibilidad de confirmar el contagio después de una exploración física o confirmar una falsa alarma, en caso de que el paciente no presente ninguna enfermedad (ver imagen Figura 2.2.7).



14 - Figura 2.2.7 – Notificaciones

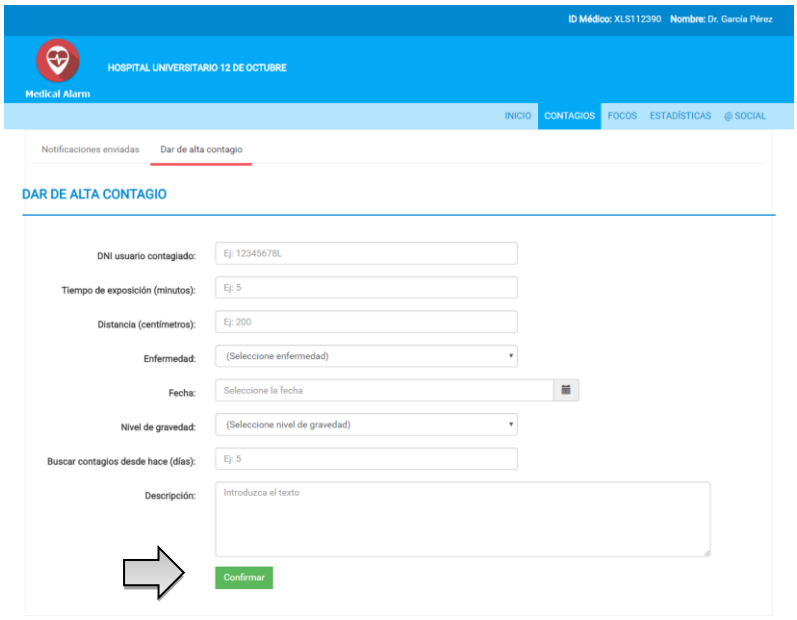
FUNCIONALIDAD 8: Confirmar contagio de un usuario

Partiendo de la imagen anterior (ver Figura 2.2.7), una vez buscadas las notificaciones del usuario, el botón rojo Contagiado sirve para confirmar que el usuario está contagiado de esa enfermedad, añadiéndola a la lista de enfermedades del usuario y cambiando el estado del mismo a ENFERMO.

FUNCIONALIDAD 9: Confirmar falsa alarma de un posible usuario contagiado

En la pestaña de Contagios, concretamente en la sub pestaña de notificaciones enviadas, además del botón Contagiado existe otro verde 'Falsa alarma' que gracias a él es posible anular el aviso de contagio y dejar SANO al usuario identificado (ver Figura 2.2.7).

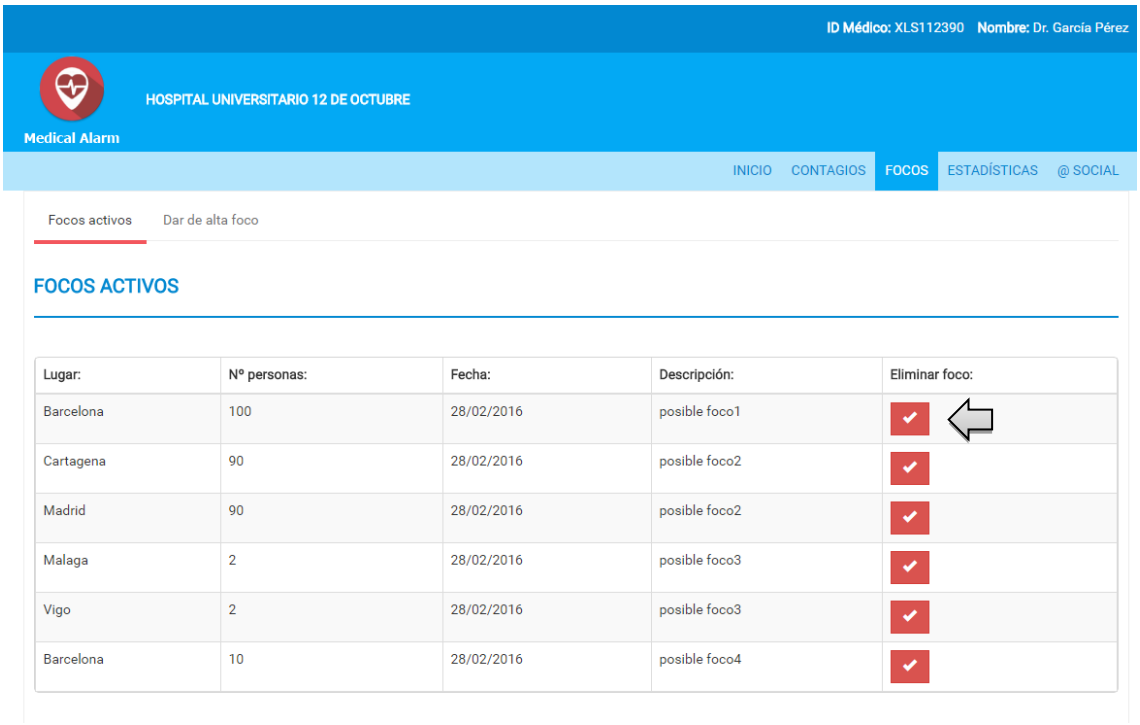
FUNCIONALIDAD 10: Dar de alta un contagio



15 - Figura 2.2.8 – Alta contagio

Otra de las funcionalidades más relevantes de la página web es el alta de un contagio. Dentro de la pestaña Contagios y la sub pestaña Dar de alta contagio, se alberga un formulario con una serie de campos para rellenar. Una vez identificado el usuario contagiado, se mete su DNI, el tiempo de exposición, la distancia, la enfermedad, la fecha, el nivel de gravedad, los días desde los cuales se va a hacer la búsqueda y una breve descripción con la información que el médico cree oportuna. A continuación, el botón verde Confirmar lanza el algoritmo que se encarga de buscar en el sistema todos los posibles usuarios que cumplan con los parámetros introducidos, pudiendo estar potencialmente infectados y generando una noticia y una notificación para cada uno de ellos (ver Figura 2.2.8).

FUNCIONALIDAD 11: Ver focos activos



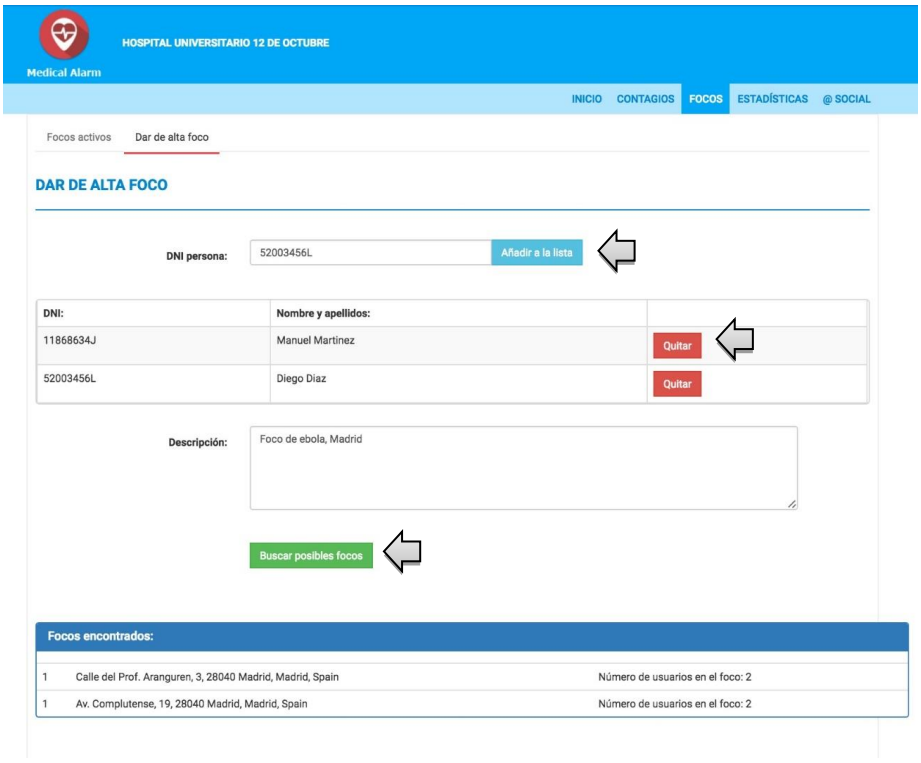
16 - Figura 2.2.9 – Focos activos

Cambiando de pestaña a Focos, dentro de Focos activos, nos encontramos con una tabla donde se muestra toda la lista de focos activos actualmente en el sistema. De cada foco es importante detallar el lugar, el número de personas involucradas, la fecha y una descripción introducida por el médico (ver Figura 2.2.9).

FUNCIONALIDAD 12: Eliminar foco

Mirando la imagen anterior (ver Figura 2.2.9) podemos ver que en la tabla de focos activos, existe un botón con un icono de "tick" que permite borrar un foco en caso de que sea oportuna su eliminación.

FUNCIONALIDAD 13: Dar de alta foco



17 - Figura 2.2.10 – Alta foco

De nuevo nos encontramos con una funcionalidad de peso en la página web, la posibilidad de encontrar el foco de una enfermedad, es decir, el lugar de origen de una enfermedad, a partir de una serie de individuos contagiados. Dentro de la pestaña de Focos y de la sub pestaña Dar de alta foco se encuentra un buscador por DNI de usuarios y un botón para añadir al usuario buscado a una lista. A continuación, todos los usuarios que han sido buscados se muestran en una tabla y cuando todos los requeridos estén en ella y se haya

rellenado una descripción del posible foco, se puede lanzar el algoritmo, pulsando en el botón verde 'Buscar posibles focos'. Cuando el algoritmo termine de cruzar todos los lugares y de obtener los resultados, se muestra una tabla con todos ellos, al igual que un mapa con una marca en el punto exacto donde puede haber sido localizado el foco de la enfermedad (ver Figura 2.2.10).

FUNCIONALIDAD 14: Eliminar usuario de la lista de foco

Partiendo de la imagen anterior (ver Figura 2.2.10), en caso de haberse confundido al introducir el usuario o debido a cualquier otro error, existe un botón rojo Quitar para deshacer la acción de añadir un usuario a la lista de usuarios antes de lanzar la búsqueda de un foco.

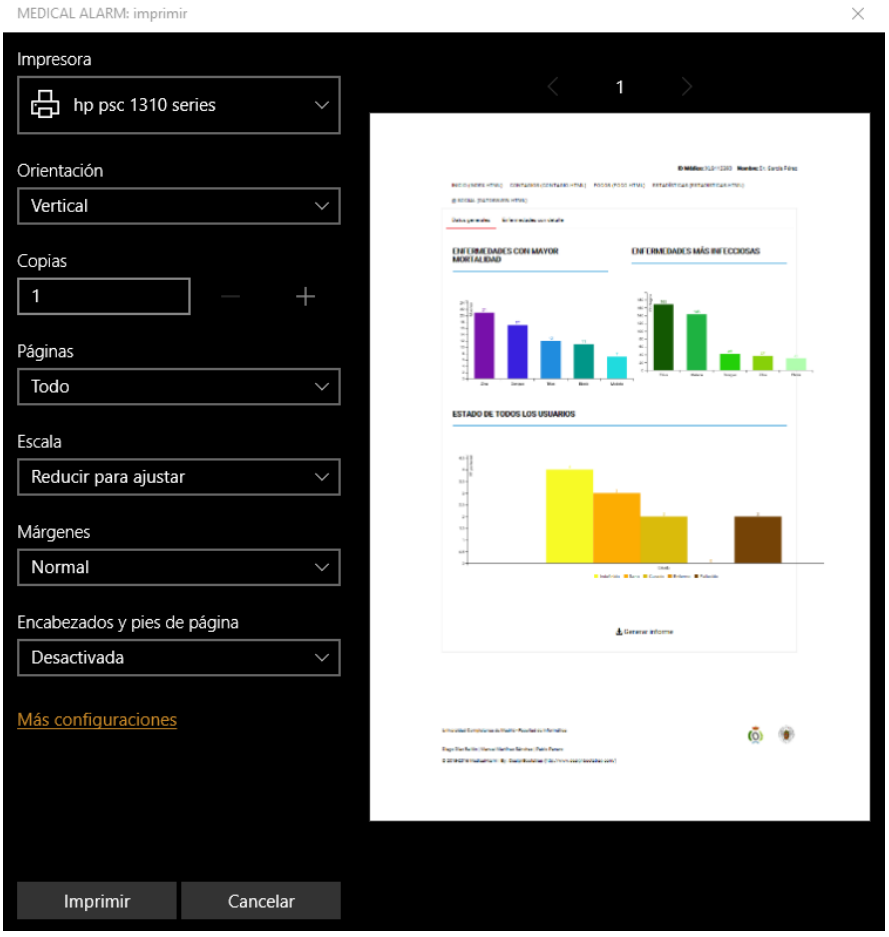
FUNCIONALIDAD 15: Visualizar datos generales

18 - Figura 2.2.11 – Estadísticas

En la pestaña de Estadísticas, se hace un resumen de todos los datos que el sistema obtiene y, mediante análisis y diferentes técnicas Big Data, se muestran a través de gráficas diferentes datos de interés. En la sub pestaña Datos generales, nada más acceder a ella, se muestran tres gráficas. Las dos primeras son un ranking del top 5 de enfermedades más contagiosas y enfermedades con mayor mortalidad, y la tercera gráfica muestra el estado de todos los pacientes, a modo de control en tiempo real del estado del sistema (ver Figura 2.2.11).

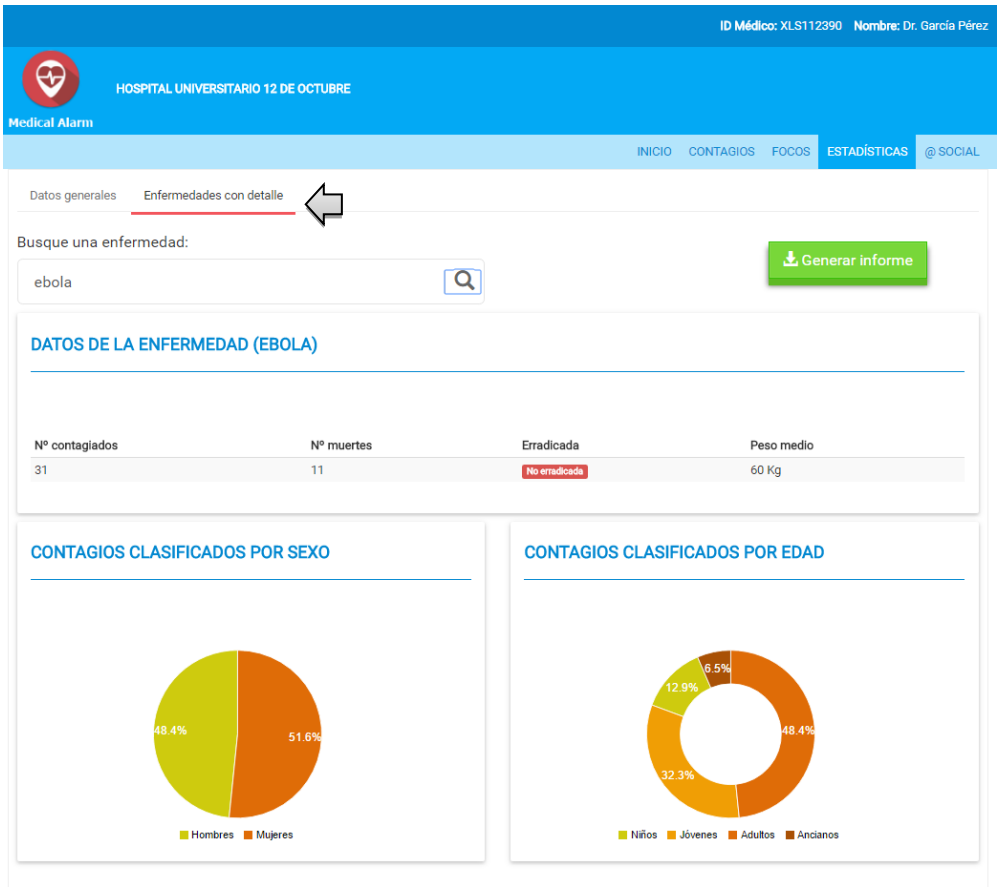
FUNCIONALIDAD 16: Generar informe datos generales

En la imagen anterior (ver Figura 2.2.11), en la parte inferior se puede apreciar un botón verde con relieve, con la función de generar un informe en formato pdf u otro tipo para comodidad del médico. Se exportarán las tres gráficas con todos los datos en el instante en el que el botón es pulsado (ver Figura 2.2.12).



19 - Figura 2.2.12 – Generar informe

FUNCIONALIDAD 17: Visualizar datos de una enfermedad

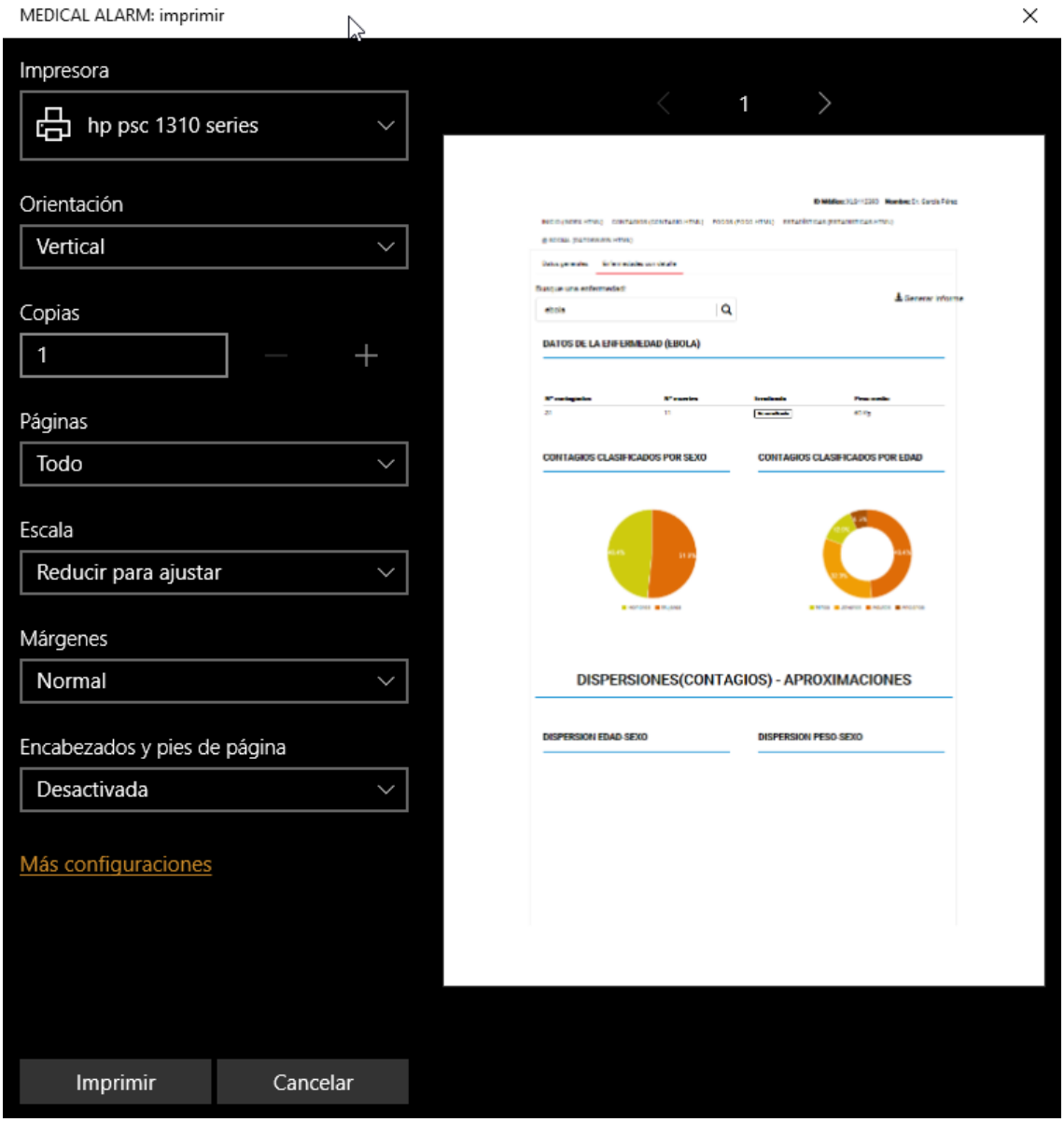


20 - Figura 2.2.13 – Datos enfermedad

Si se pulsa sobre la pestaña Estadísticas y la sub pestaña Enfermedades con detalle, aparece en primer lugar un buscador que, a partir del nombre de la enfermedad, nos proporciona gran cantidad de información relacionada con ella. En primer lugar, se muestra una tabla con el número de personas contagiadas por esa enfermedad, las muertes que ha producido, si está erradicada o no y el peso medio al que afecta esa enfermedad. A continuación se muestran una serie de gráficas, primero dos con forma de sector circular donde se puede ver una clasificación por grupos de edades y sexo según ha afectado la enfermedad infecciosa y, por último, se infieren una serie de gráficas de dispersión en las que se relacionan las tres medidas más importantes que el sistema posee, que son: el sexo, la edad y el peso. Gracias a todo esto, se puede decir que el sistema es inteligente y de manera automática proporciona al médico datos de gran utilidad para posibles estudios futuros y prevenciones.

FUNCIONALIDAD 18: Generar informe datos enfermedad

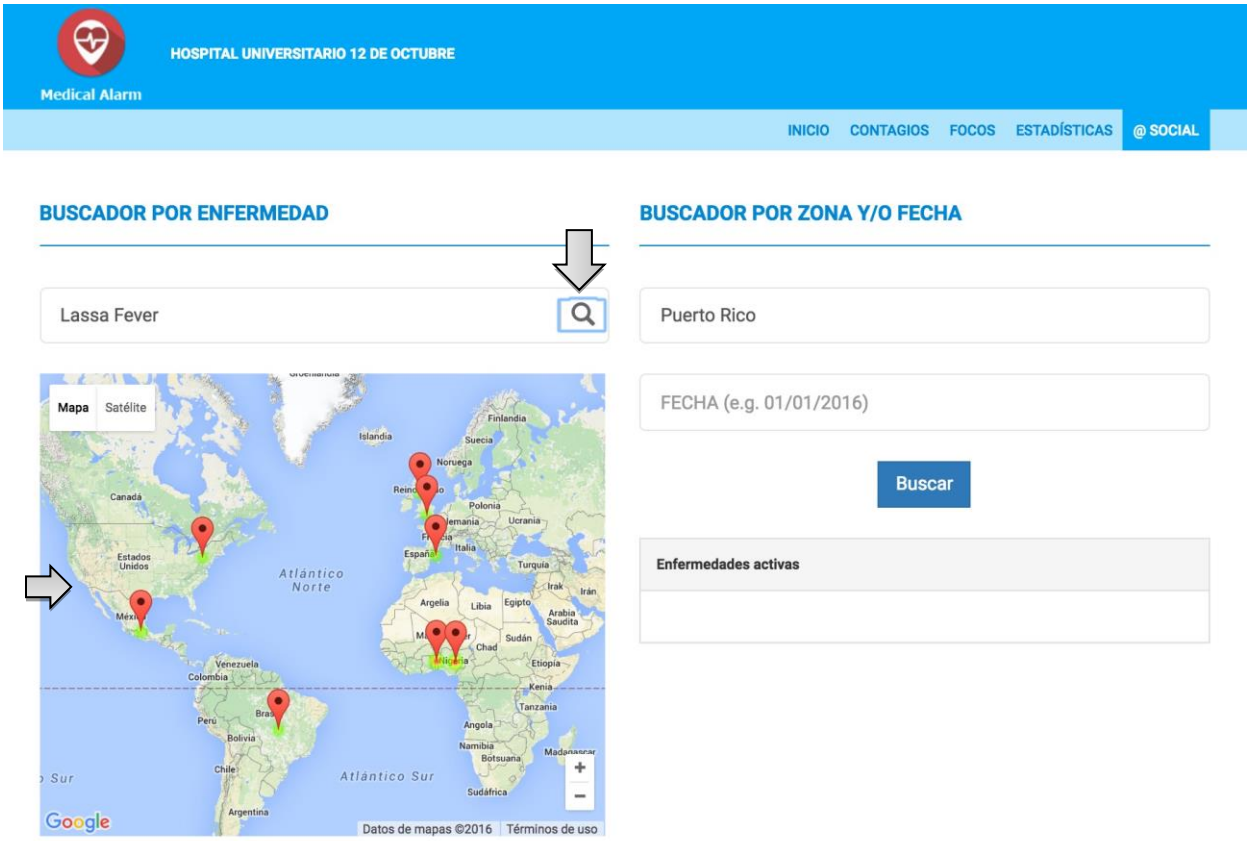
Fijándose en la funcionalidad anterior, parece extraño que no exista la posibilidad de guardar toda la información obtenida de cada enfermedad registrada, por lo que en la vista de la enfermedad, dentro de las estadísticas, existe otro botón verde con relieve que permite exportar todos los datos y generar un informe en pdf con todas las gráficas y tablas de la enfermedad encontrada en el buscador (ver Figura 2.2.14).



21 - Figura 2.2.14 – Generar informe enfermedad

FUNCIONALIDAD 19: Consultar mapa de calor global por enfermedad

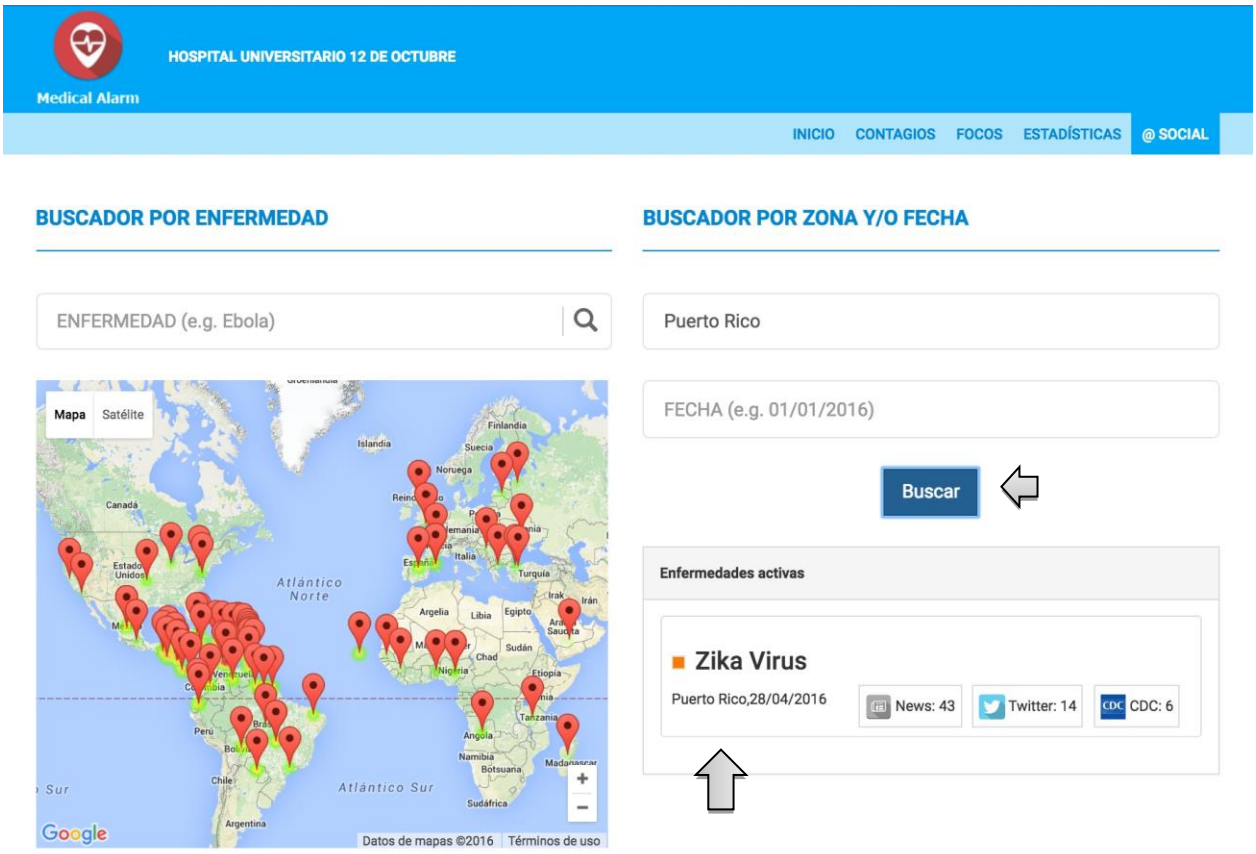
Además del mapa de calor mostrado en la página principal en la pestaña "*@Social*" existe un mapa de calor que se completa con la presencia de enfermedades a nivel global. Según los datos obtenidos de la web con el recolector de datos. Si no se introduce el nombre de ninguna enfermedad (ver Figura 2.2.15) y se pulsa el botón de buscar se obtendrá como resultado el mapa de calor con todas las enfermedades almacenadas en el sistema.



22 - Figura 2.2.15 – Mapa calor

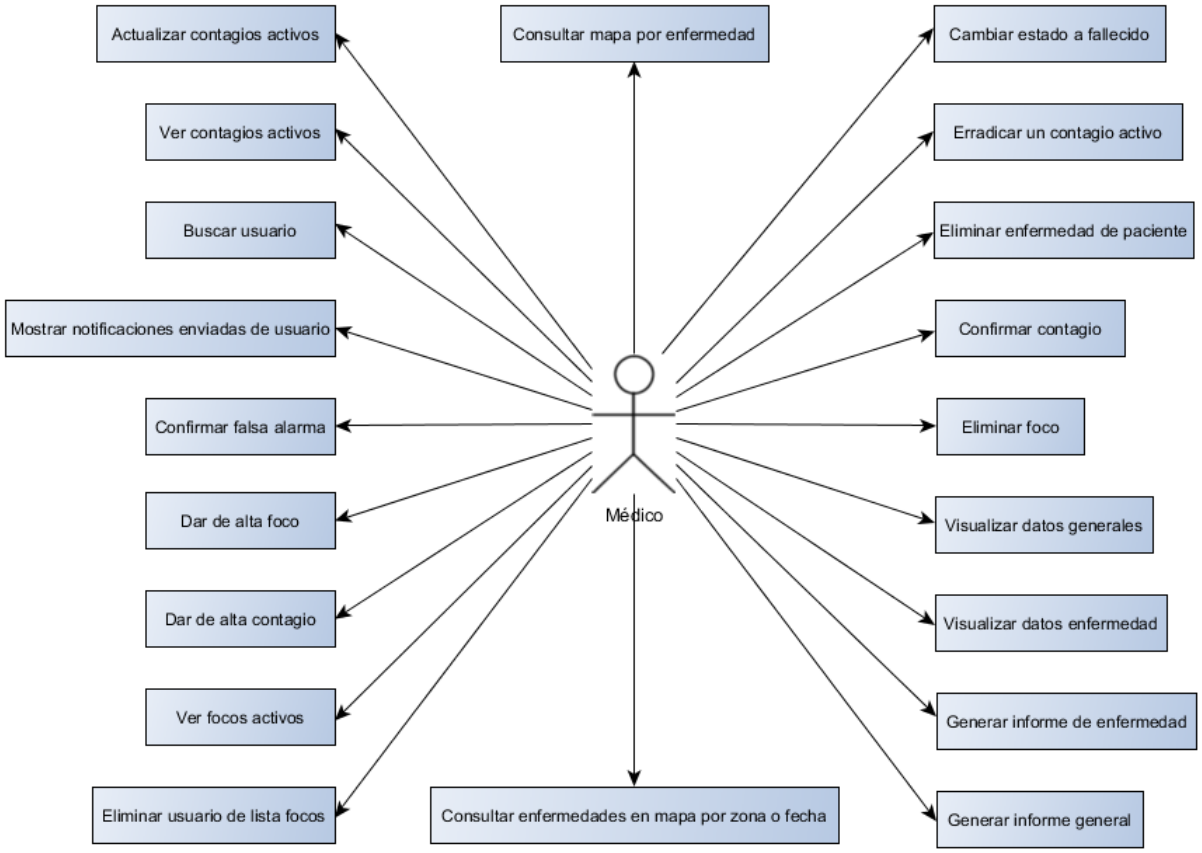
FUNCIONALIDAD 20: Consultar enfermedades globales por zona o fecha

En este caso, no en mapa de calor, pero sí en forma de lista con sus atributos, se puede realizar una consulta con dos parámetros zona (e.g. Puerto Rico, España) de la cual se quiere ver la lista de enfermedades y fecha, para poder filtrar la fecha en la cual debían estar activas las enfermedades. Se pueden realizar búsquedas con ambos, con uno, o con ninguno de los dos parámetros (ver Figura 2.2.16).



23 - Figura 2.2.16 – Consultar enfermedades globales

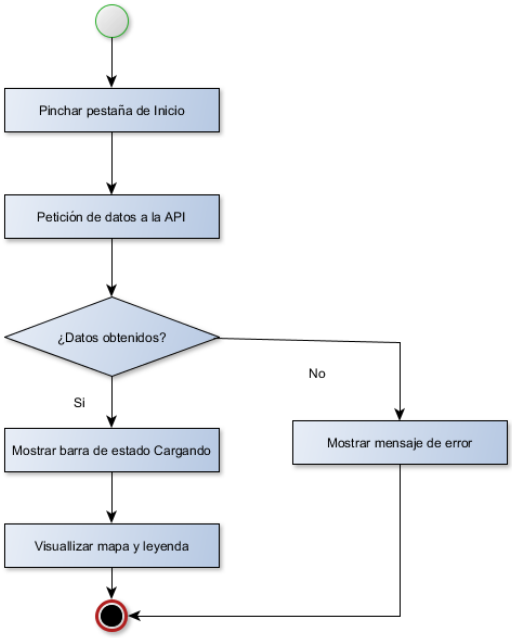
A modo de resumen, hemos realizado los diagramas de casos de uso para que se pueda apreciar de una manera más clara el conjunto de funcionalidades que la página web ofrece. En el diagrama, el actor es el médico pues es el usuario de la página web.



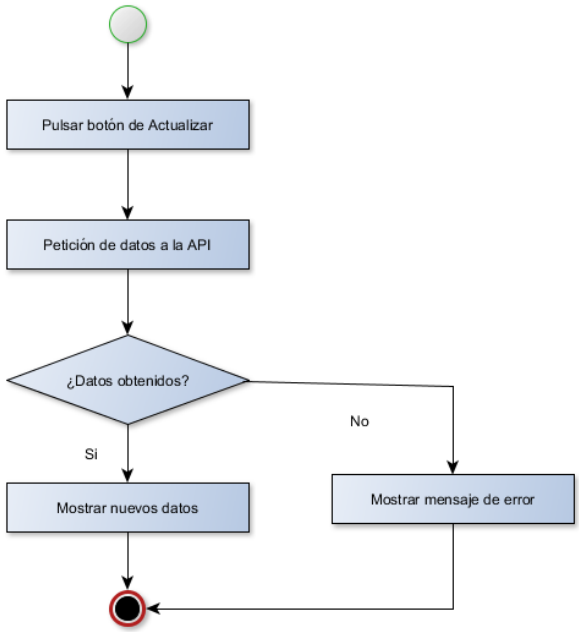
24 - Figura 2.2.17 – Casos de uso web

2.2.1 Diagramas de actividades

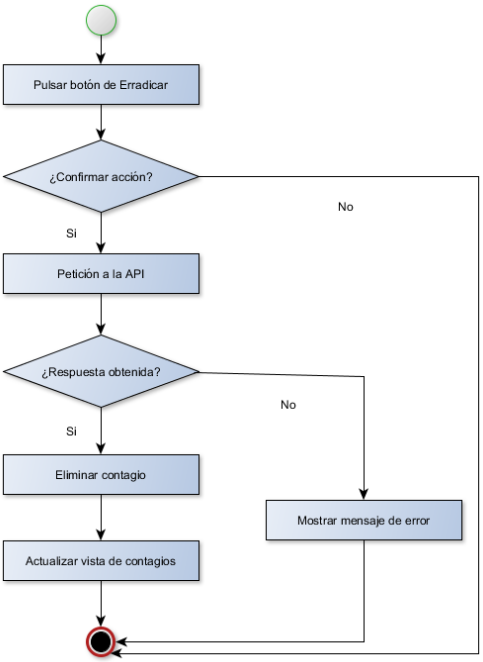
Ver contagios activos:



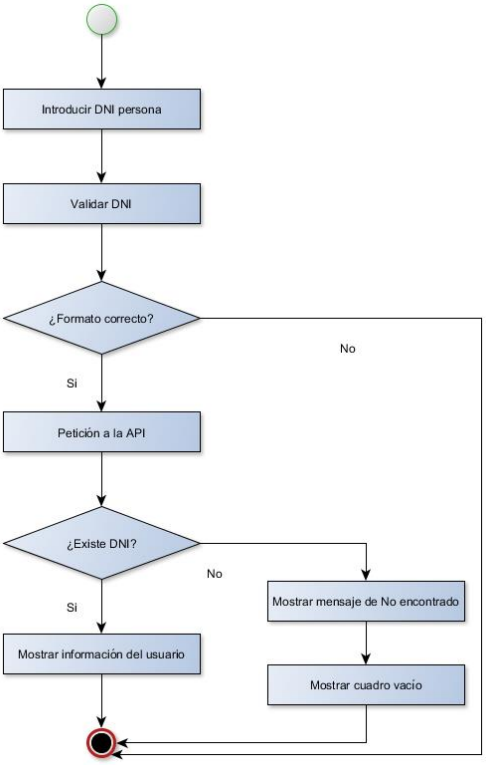
Actualizar contagios activos:



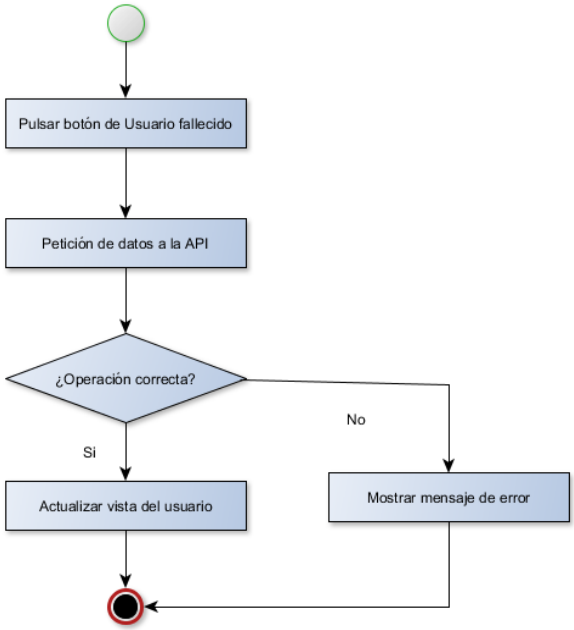
Erradicar contagio activo:



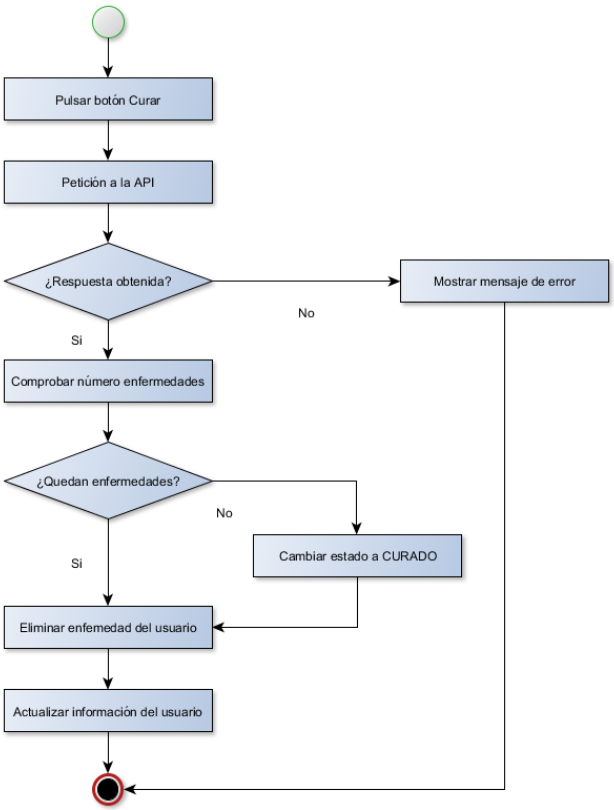
Buscar un usuario:



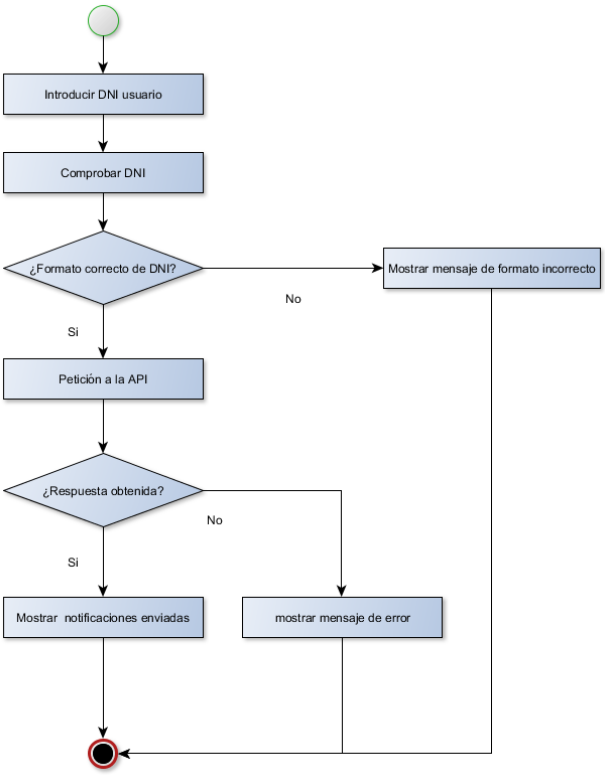
Cambiar estado a fallecido:



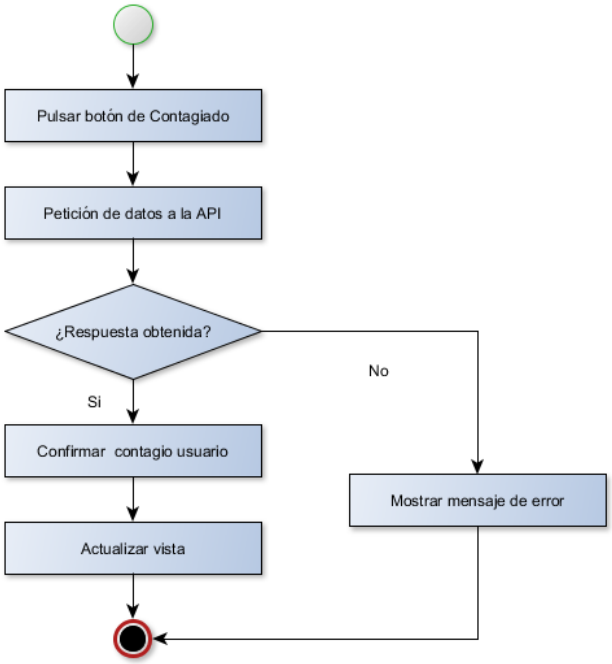
Eliminar enfermedad:



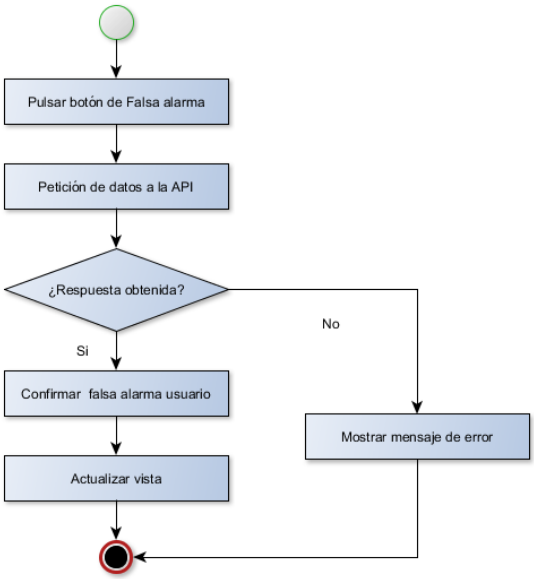
Mostrar notificaciones enviadas:



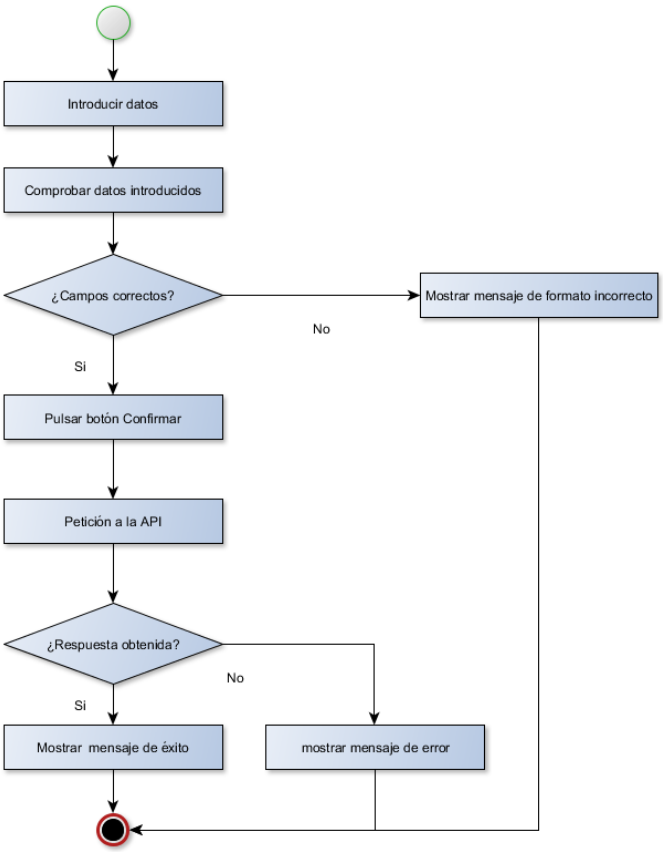
Confirmar contagio de un usuario:



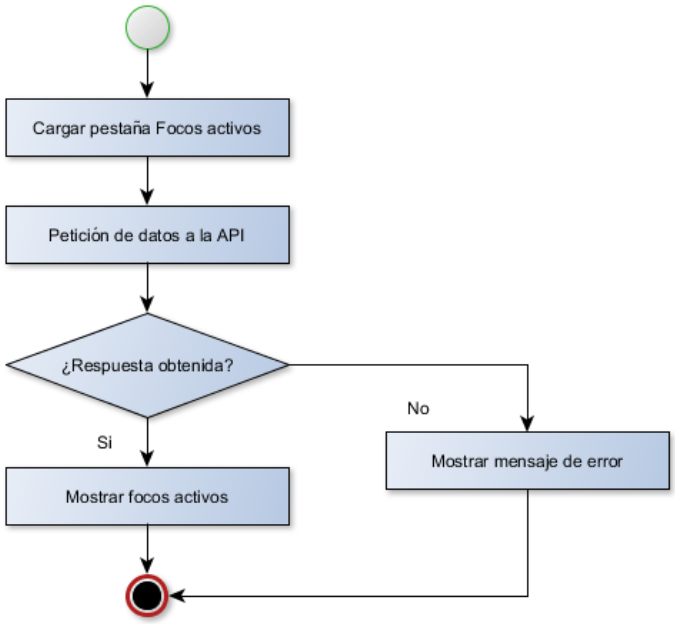
Confirmar falsa alarma de usuario:



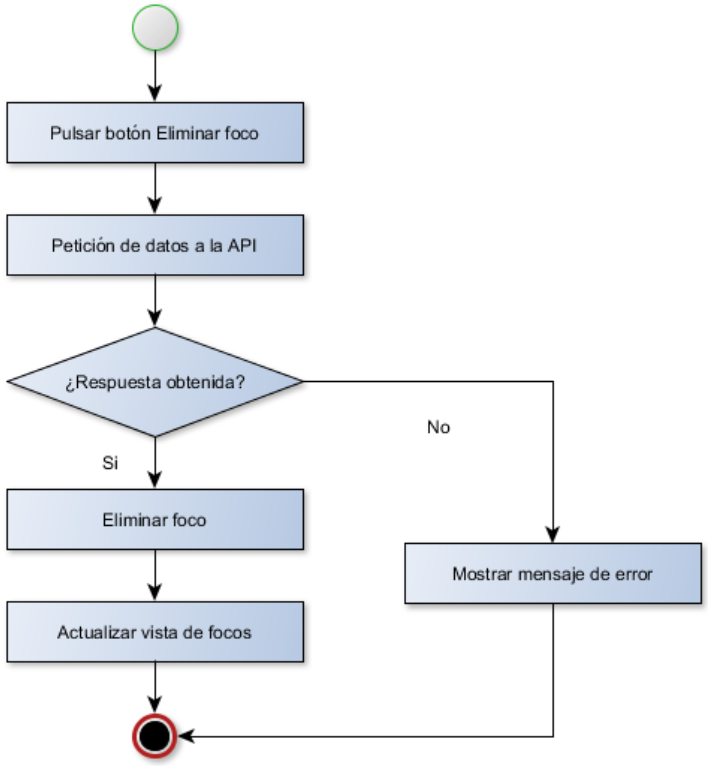
Dar de alta contagio:



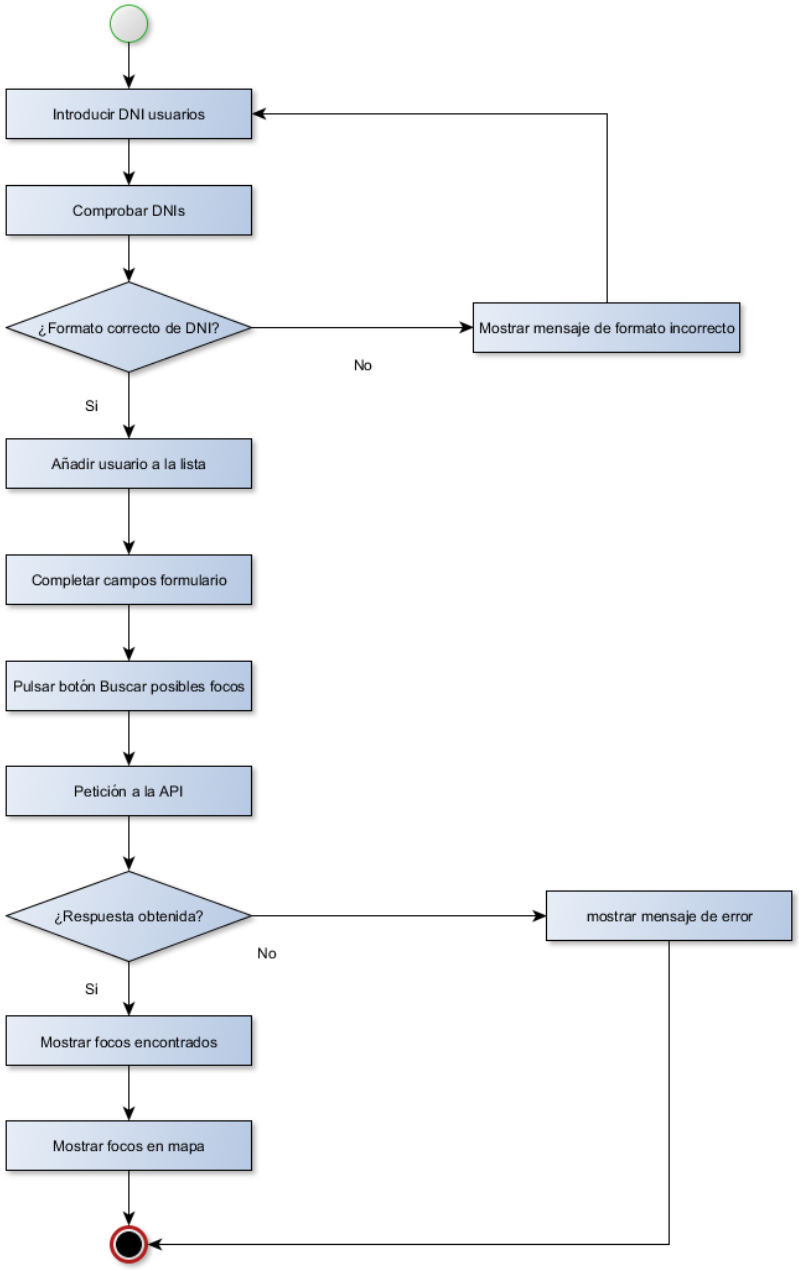
Ver focos activos:



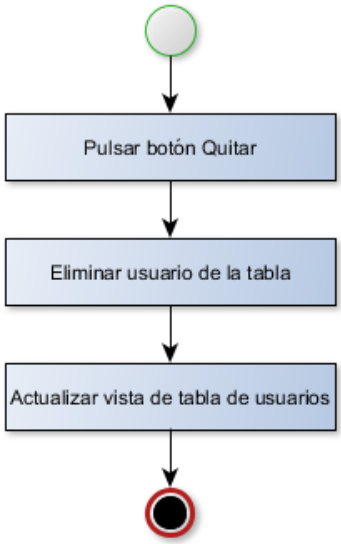
Eliminar foco:



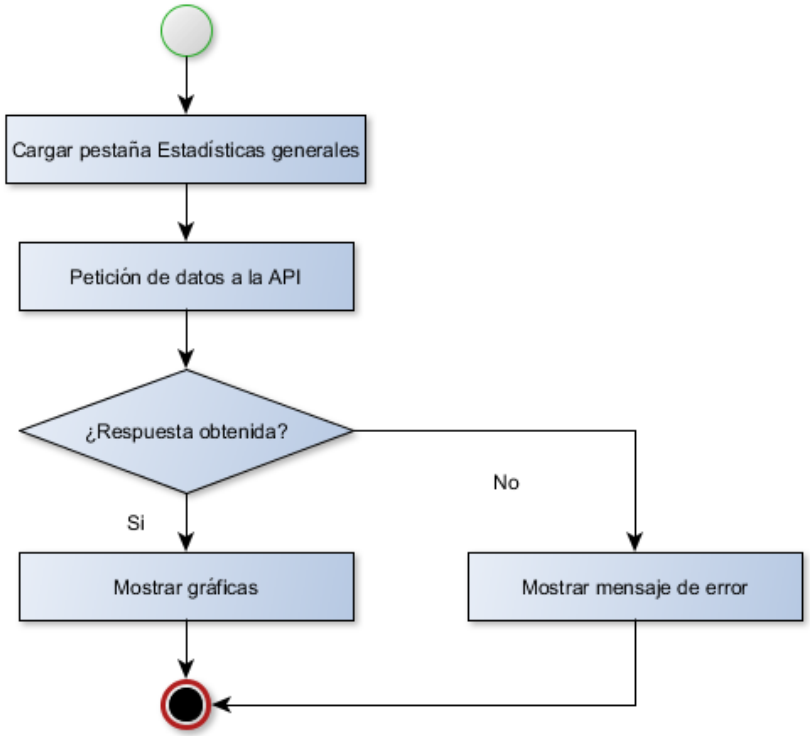
Dar de alta foco:



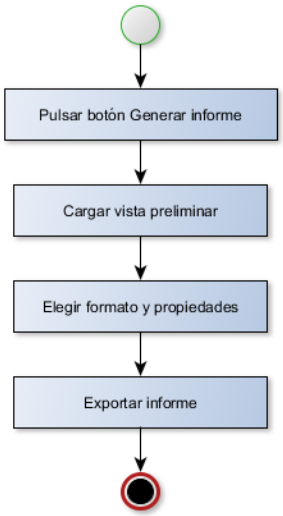
Eliminar usuario de lista de foco:



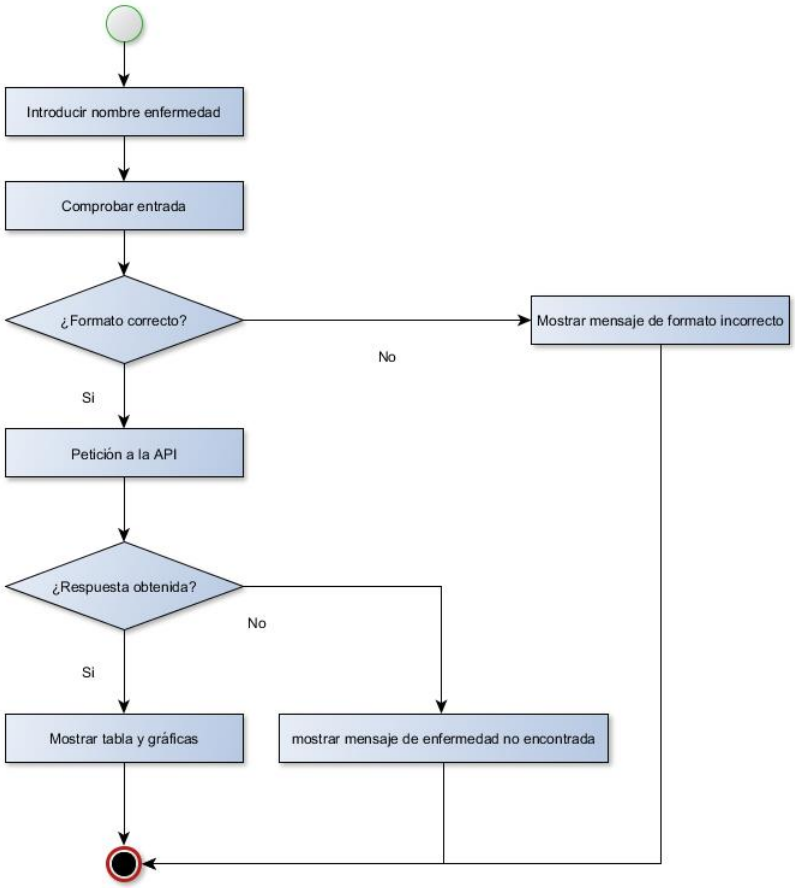
Visualizar estadísticas generales:



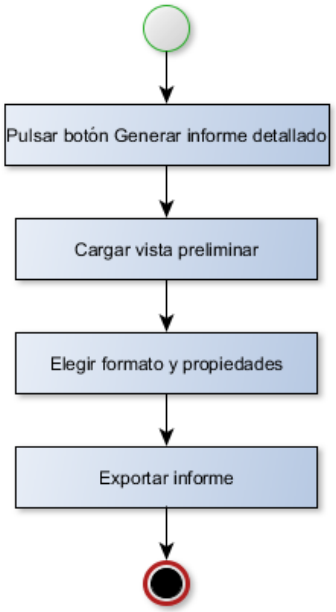
Generar informe general:



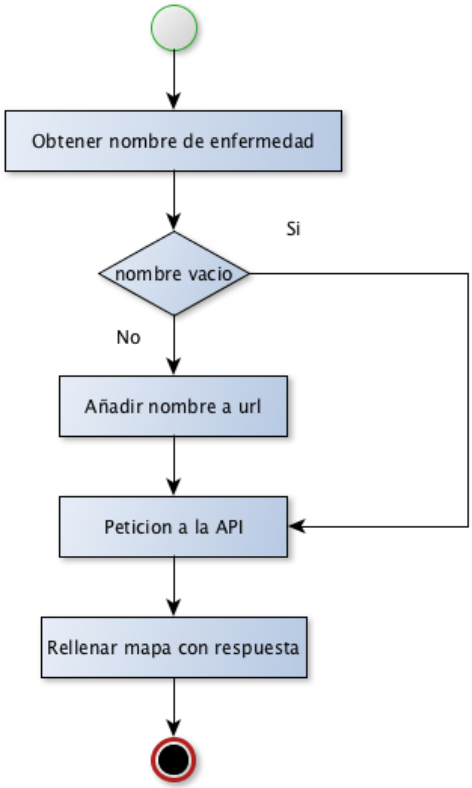
Visualizar estadísticas enfermedad:



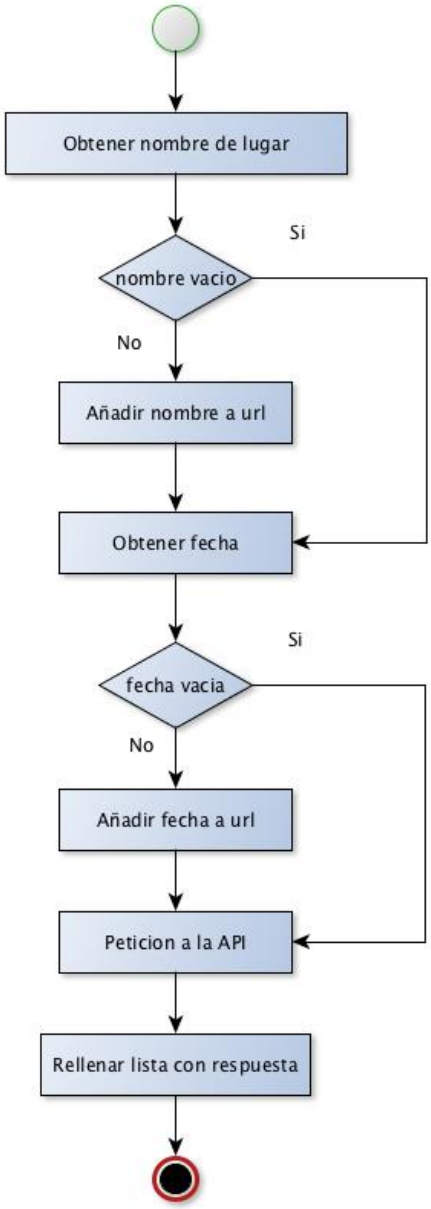
Generar informe enfermedad:



Consultar mapa de calor global por enfermedad:



Consultar lista de enfermedades globales por zona y fecha:



2.3 Funcionalidad API y recolector de datos

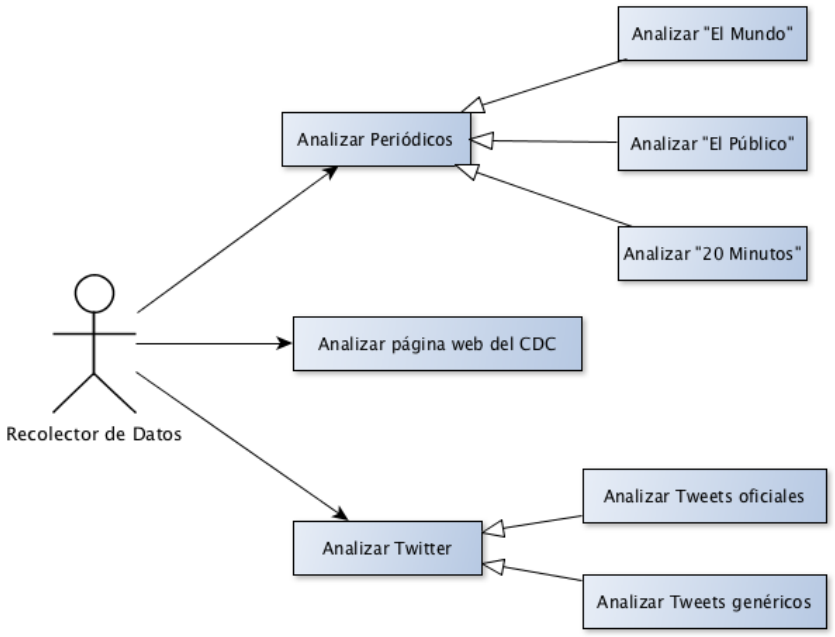
2.3.1 Recolector de datos sociales

Para poder llevar a cabo un análisis de los datos que son generados en redes sociales como Twitter, en páginas web de organizaciones oficiales como el CDC y en los periódicos online, se hace uso de varios programas. La función principal de los datos obtenidos de la Web es comprobar la presencia de enfermedades en las distintas localidades del planeta, y además medir la "preocupación" o "concienciación" que tiene la población de una localidad geográfica sobre una enfermedad.

En primer lugar se cuenta con el programa que analizada las secciones correspondientes a salud de "El mundo", "El público" y "20 minutos". Existe un programa que se ejecuta una vez al día para cada uno de ellos, analizando las páginas en busca de noticias que mencionen alguna de las enfermedades actualmente activas en idioma castellano o inglés. En caso de encontrar alguna, analiza el texto en busca de localizaciones geográficas, lenguaje y sentimiento (positivo, neutro o negativo), para poder así dar un valor más adecuado al contenido y la "preocupación" que refleja, actualizando con ello las enfermedades correspondientes y los puntos geográficos en los cuales se encuentran.

Además hay otro programa que analiza la sección de alertas de la página del CDC (Centro de Control de Enfermedades). Al igual que con las noticias en el apartado anterior, se ejecuta con una periodicidad diaria y actualiza la base de datos con la información recogida sobre enfermedades y su disposición geográfica.

Para analizar los datos producidos en la red social Twitter se hace uso de dos programas que están en continuo funcionamiento. El primero de ellos está pendiente de los tweets producidos por cuentas oficiales, como puede ser "sanidadgob" (correspondiente al Ministerio de Sanidad del Gobierno Español). El segundo programa analiza en tiempo real todos los tweets producidos que contengan en su texto alguna de las enfermedades actualmente activas, en idioma castellano o inglés. Para ambos casos se analiza el texto en busca de localizaciones geográficas, lenguaje y sentimiento, y se actualiza la base de datos con las enfermedades correspondientes y sus puntos geográficos.



25 - Figura 2.3.1.1 - Recolector datos sociales

2.3.2 API (Interfaz para la programación de aplicaciones)

Las APIs son el punto de entrada para las aplicaciones móviles y la página web, haciendo posible la abstracción de su funcionalidad y la interacción con las bases de datos para guardar y solicitar datos.

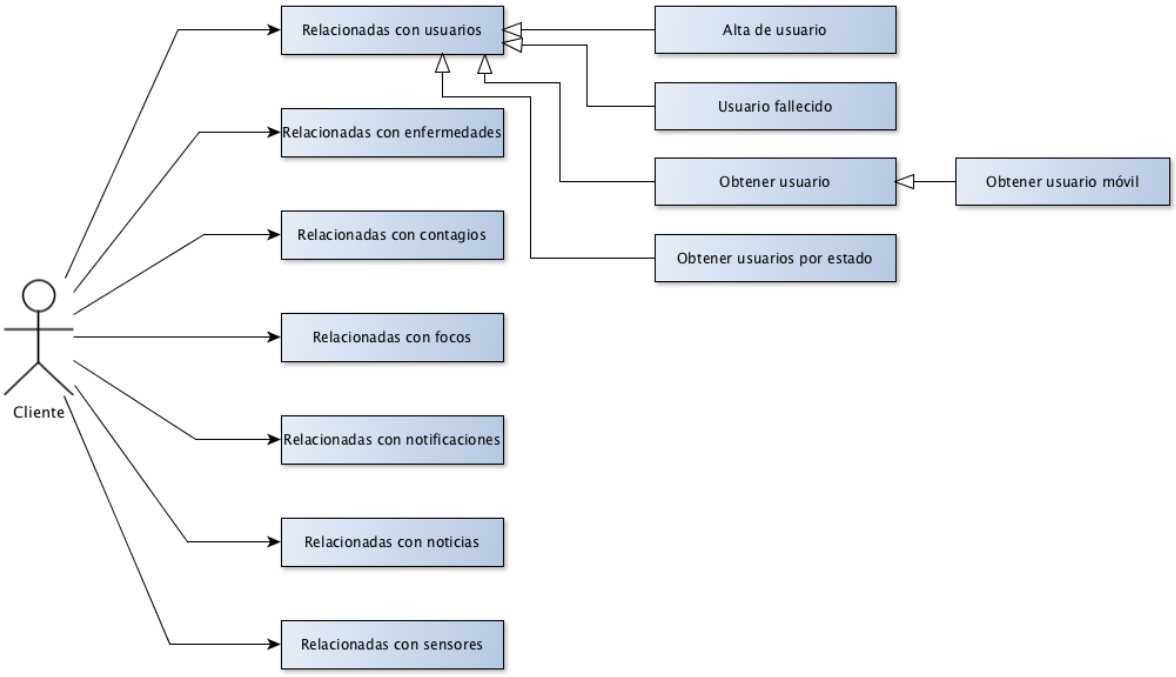
Este proyecto cuenta con dos APIs que consultan bases de datos diferentes. La primera API permite interactuar con todos los datos relacionados con usuarios registrados en el sistema (enfermedades, notificaciones, noticias, contagios y focos). Y la segunda API permite interactuar con los datos sobre enfermedades y zonas geográficas procedentes de redes sociales, periódicos y páginas web oficiales. Para facilitar la comprensión del lector llamaremos API del sistema a la mencionada en primer lugar y API social a la segunda.

La API del sistema ofrece la siguiente funcionalidad:

Relacionadas con los usuarios:

- Alta de usuario: dados el nombre, los apellidos, el email, la fecha de nacimiento, el género, el peso, el documento de identificación (DNI/NIE) y la contraseña, añade el usuario a la base de datos si no existe ningún usuario con el mismo documento de identificación o con el mismo email.

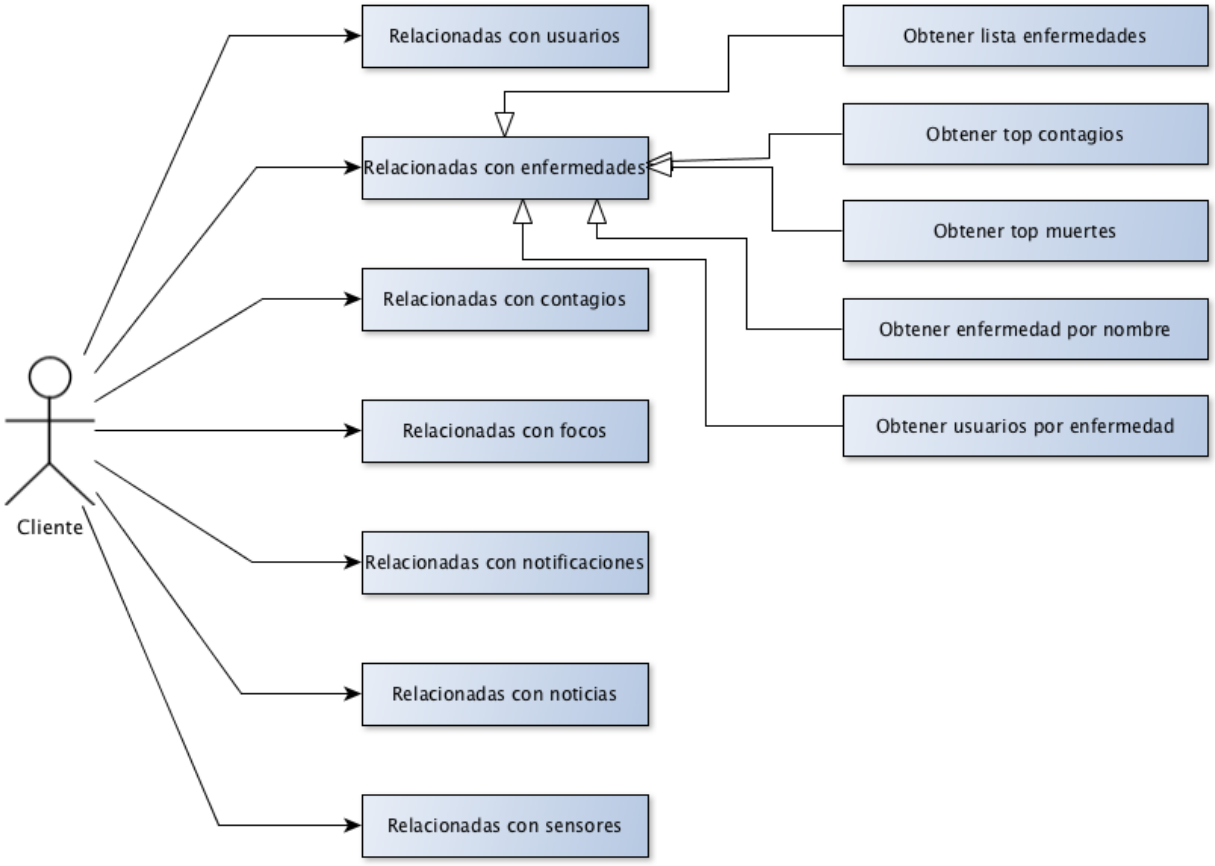
- Edición del estado del usuario a fallecido: en el caso de que un usuario fallezca, se cambia su estado a "fallecido" y se aumenta en una unidad la cantidad de muertes provocadas por las enfermedades a las que pertenecen los contagios activos de dicho usuario.
- Obtención de los datos de un usuario según documento de identificación y contraseña para la comprobación al entrar en la aplicación móvil.
- Obtención de un usuario según el documento de identificación. Si dicho usuario existe, se obtienen sus datos sociodemográficos y los contagios activos con los datos de la enfermedad a la que pertenecen.
- Obtención de la cantidad de usuarios que se encuentran en cada estado (indefinido, sano, curado, enfermo, fallecido).



26 - Figura 2.3.2.1 – API usuarios

Relacionadas con enfermedades:

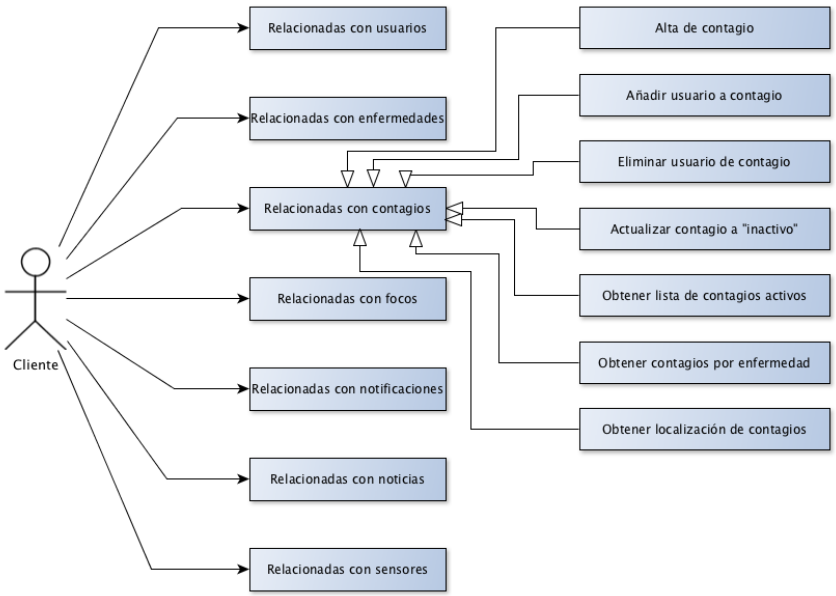
- Obtención de la lista de enfermedades.
- Obtención de la cantidad deseada de las enfermedades con mayor número de usuarios contagiados.
- Obtención de la cantidad deseada de las enfermedades con mayor número de muertes causadas.
- Obtención de los datos de una enfermedad según el nombre.
- Obtención de la lista de usuarios afectados por una enfermedad con sus datos de género, edad, y peso.



27 - Figura 2.3.2.2 – API enfermedades

Relacionadas con los contagios:

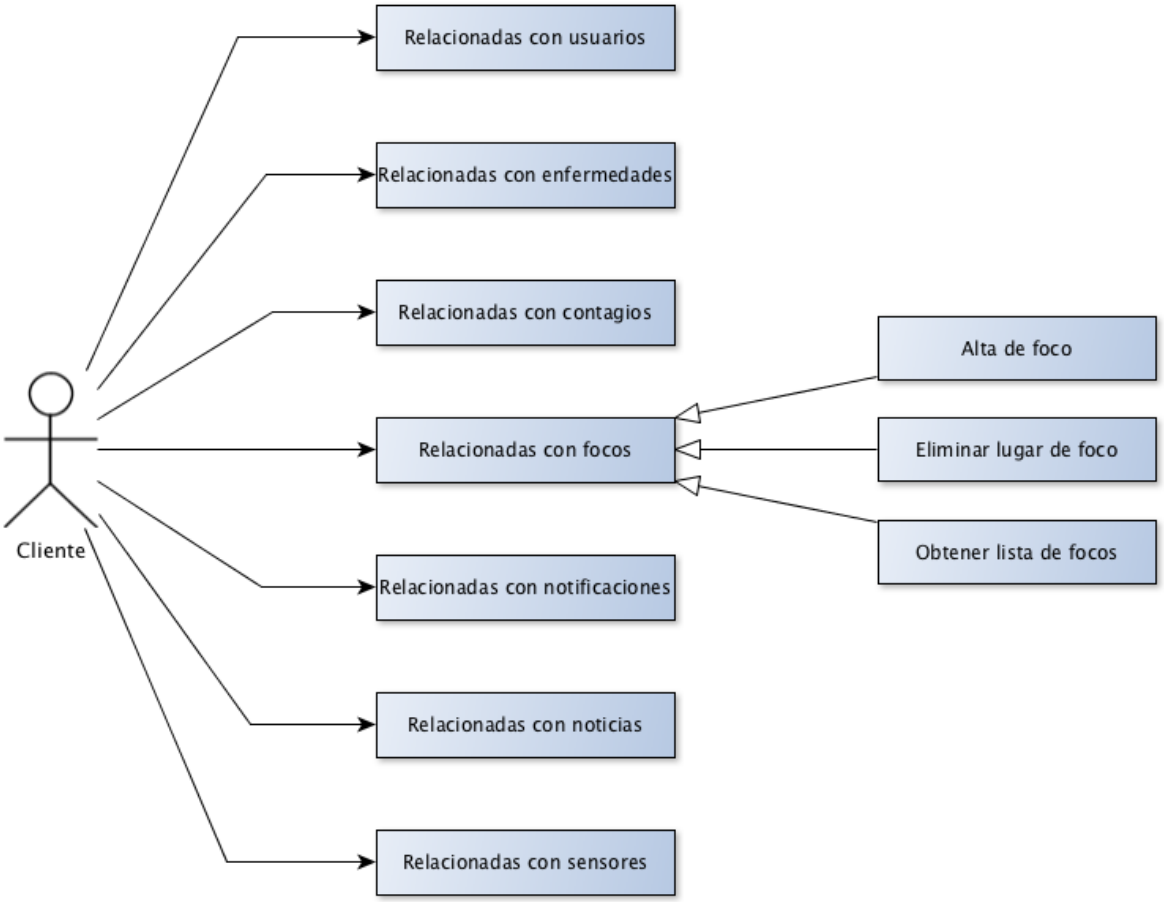
- Dado un usuario inicial, una distancia, una ventana de tiempo hacia el pasado, un tiempo de exposición y una descripción, se analiza en busca de los posibles usuarios contagiados añadiendo una notificación para los posibles afectados.
- Dado un usuario y un contagio, se añade dicho usuario a la lista de usuarios pertenecientes a dicho contagio. Además, se actualizan los datos de la enfermedad a la que pertenece dicho contagio: se añade una unidad a la cantidad de personas infectadas en el rango de edad al que pertenece el usuario, se actualiza peso medio al que afecta dicha enfermedad y se añade una unidad a la cantidad de personas contagiadas del mismo género que el usuario.
- Dado un usuario y un contagio, se elimina dicho usuario de la lista de usuarios pertenecientes a dicho contagio. Además si el usuario no tiene más contagios activos se cambia su estado a "curado".
- Dado un contagio se actualiza el estado de dicho contagio a "inactivo".
- Obtención de la lista de todos contagios activos.
- Obtención de la lista de contagios que pertenecen a una enfermedad, según el nombre, y la lista de usuarios que pertenecen a cada uno de dichos contagios.
- Obtención de los puntos geográficos, latitud y longitud, donde se encuentran los contagios activos.



28 - Figura 2.3.2.3 – API contagios

Relacionadas con los focos:

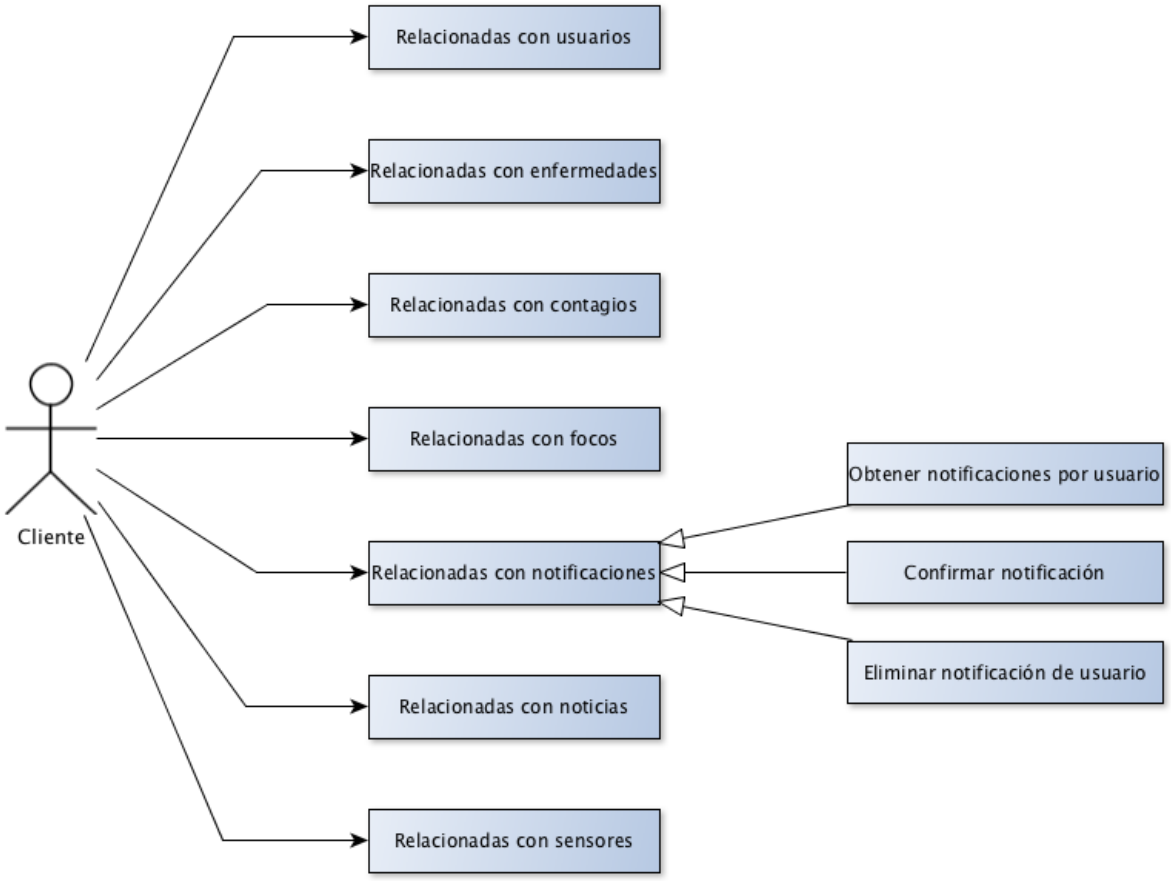
- Dada una lista de identificadores de usuario (DNI/NIE) y una enfermedad, se obtienen los posibles puntos donde se pudieron contagiar (focos) tanto todos los usuarios de la lista, como un grupo reducido de ellos. Además, se añade un foco a la enfermedad dada, y un lugar por cada punto encontrado a dicho foco.
- Dado un foco y un lugar, se elimina de la lista de lugares de dicho foco el lugar dado.
- Obtención de la lista de focos activos con sus respectivos lugares.



29 - Figura 2.3.2.4 – API focos

Relacionadas con las notificaciones:

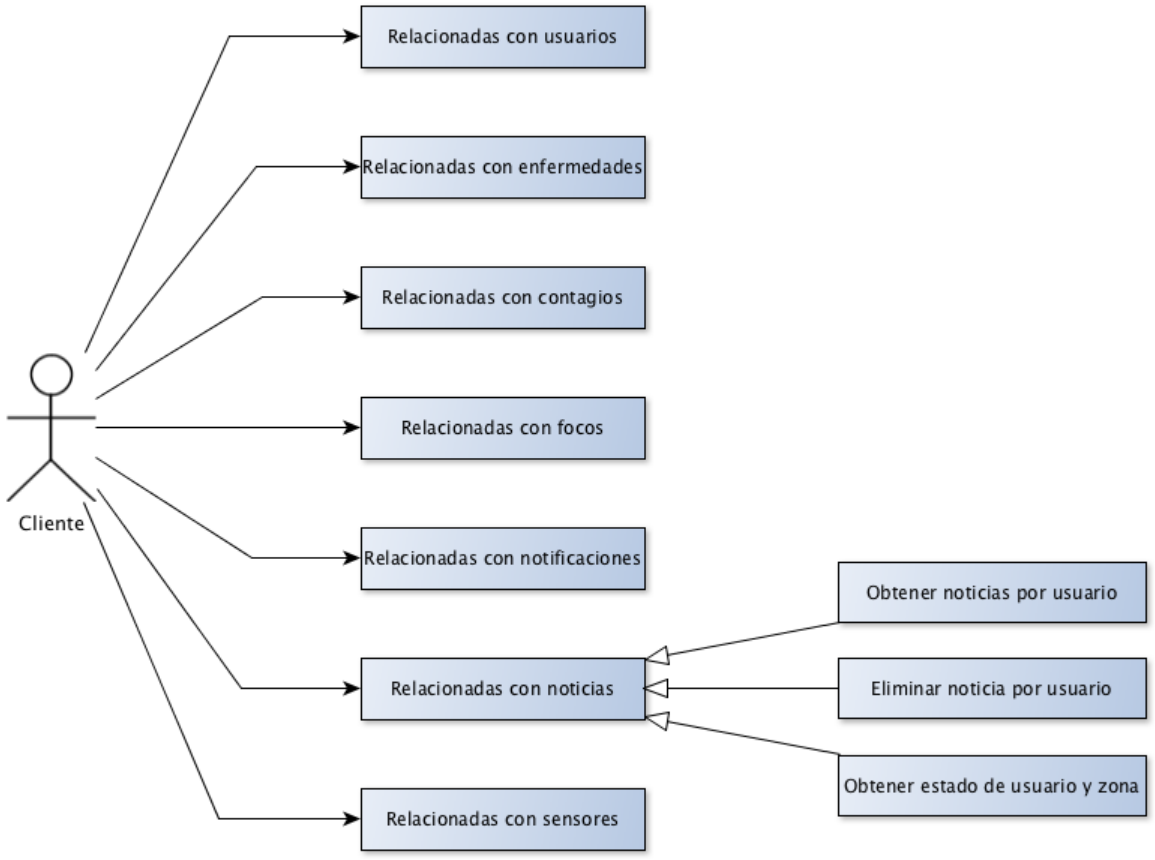
- Dado el documento de identificación de un usuario, se obtienen todas las notificaciones pendientes, es decir, las posibles enfermedades que han sido contraídas por dicho usuario y que ha detectado el sistema.
- Dado un usuario y una notificación, se verifica que dicho usuario ha asistido a la revisión médica con respecto a un posible contagio. Si el usuario da positivo refiriéndose a la enfermedad del contagio, se añade dicho usuario a la lista de usuarios contagiados (actualizando apropiadamente los datos de la enfermedad).
- Dado un usuario y una notificación, se elimina dicha notificación del sistema. Además, se actualiza el estado del usuario a "sano" o "curado" según corresponda.



30 - Figura 2.3.2.5 – API notificaciones

Relacionadas con las noticias:

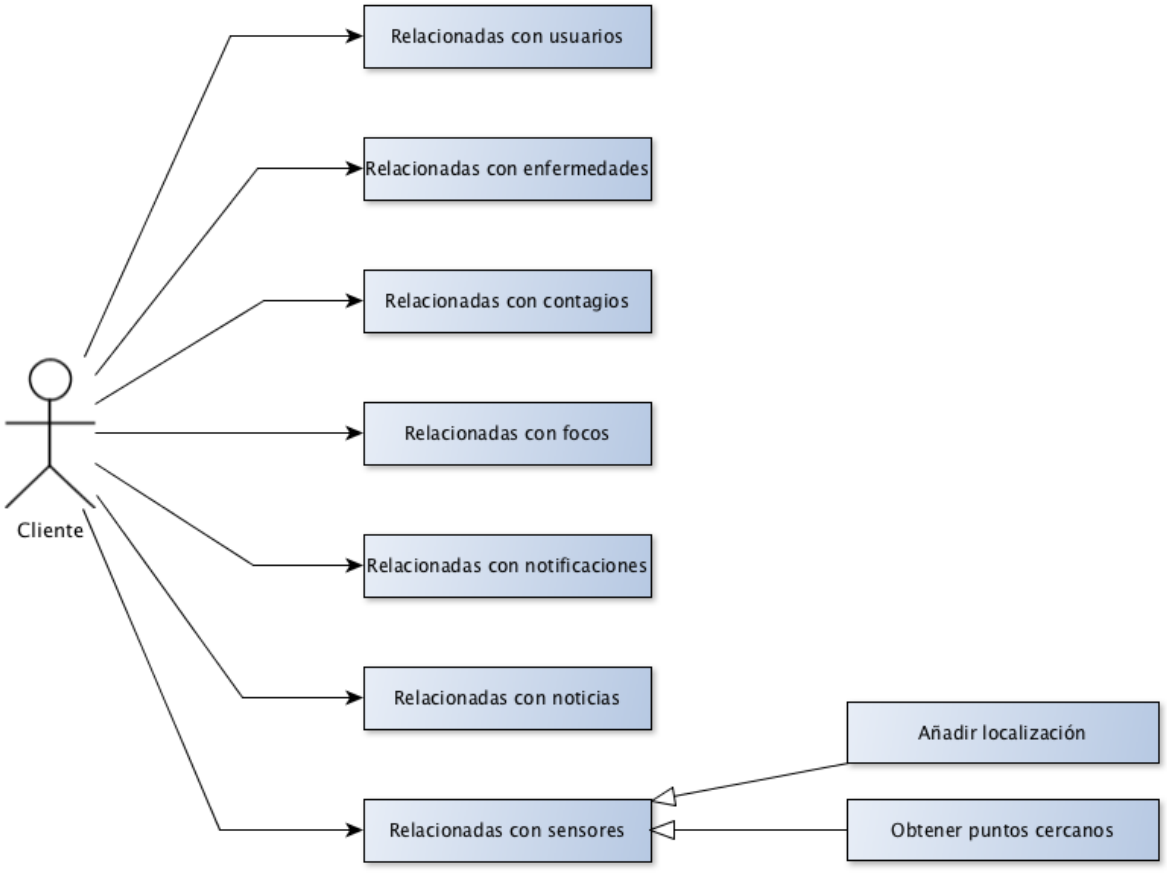
- Dado un usuario se obtiene la lista de todas las noticias que dicho usuario no ha visto.
- Dado un usuario y una noticia se elimina dicha noticia de la lista del usuario.
- Dado un usuario y unas coordenadas geográficas, latitud y longitud, se obtienen el estado del usuario y el estado de la zona en la que se encuentra el usuario.



31 - Figura 2.3.2.6 – API noticias

Relacionadas con los datos de los sensores de la aplicación móvil:

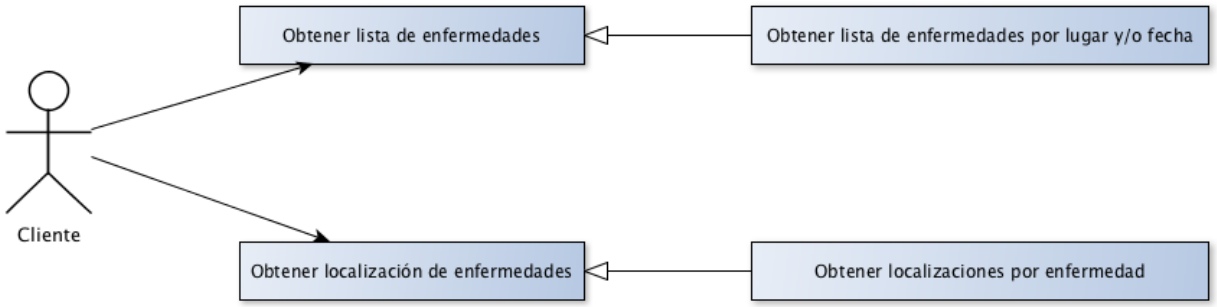
- Dado el documento de identificación de un usuario y los datos de los sensores (latitud, longitud, magnetómetro, acelerómetro, sensor de luz, sensor de batería, y el instante de tiempo en el que se obtuvieron dichos datos), se añaden los datos de los sensores al usuario, siempre que éste exista.
- Dadas unas coordenadas geográficas, latitud y longitud, y una distancia, se obtienen los datos de aquellos sensores cercanos (a una distancia menor o igual a la dada) a dicho punto en el espacio.



32 - Figura 2.3.2.7 – API sensores

La API social ofrece las siguientes funcionalidades:

- Obtener la lista de enfermedades activas junto con el lugar donde están presentes y la cantidad de veces que han sido mencionadas en twitter, periódicos y en la página web del CDC.
- Obtener la lista de enfermedades activas en un lugar y/o en una fecha en concreto, al igual que antes con la información respectiva sobre las menciones en la WWW.
- Obtener la lista de localizaciones geográficas y el peso de cada punto de las enfermedades activas. Así como los respectivos centros de los focos.
- Obtener la lista de localizaciones geográficas y el peso de cada punto para una enfermedad en concreto. Así como todos los respectivos centros de los focos.



33 - Figura 2.3.2.8 – API social

Capítulo 3. Tecnologías empleadas

A continuación se detallan las tecnologías usadas para cada módulo del proyecto. Para el desarrollo del sistema se utilizara *Git (GitHub)*[2] como sistema de gestión de versiones en un servidor.

3.1 Aplicación Android

Para el desarrollo de la aplicación se utiliza Java y el SDK de Android. El IDE elegido para esta parte es Android Studio, debido a su baja curva de aprendizaje y funcionalidades que ayudan al desarrollo y codificación. Además se utiliza *Material Design* como estándar para la interfaz de usuario.

Se ha optado por utilizar la plataforma de Android debido a ser ésta de código abierto, tener una comunidad más activa de desarrolladores y por carecer de dispositivos con otro sistema operativo para su testing.

3.2 Aplicación web

Para desarrollar este módulo se utilizan tecnologías como *HTML5*, *CSS3*, y *JavaScript*, con ayuda de Frameworks y librerías como *Bootstrap* debido a que están muy extendidas y existe una amplia documentación. Además, Bootstrap permite un diseño '*responsive*' facilitando la visualización entre distintos dispositivos.

3.3 Base de datos

Para el diseño y desarrollo de las bases de datos se utilizan tecnologías diferentes dependiendo de la naturaleza de almacenamiento de cada una.

[2] GitHub es una plataforma de desarrollo colaborativopara alojar proyectosutilizando el sistema de control de versiones Git.

En este sentido, las bases de datos relacionales están diseñadas e implementadas utilizando *MySQL* ya que es de código abierto, mientras que para las bases de datos no relacionales se utiliza *MongoDB*, dada la naturaleza no estructurada de los datos y la amplia variedad de funcionalidades que ofrece para el tratamiento de los mismos.

3.4 Servidor(es) y unidad(es) de procesamiento

En el lado del servidor, para la aplicación web se utiliza *Tomcat6 Catalina* como servidor web, en lugar de alternativas como *nginx*, ya que es el que se encuentra instalado en el servidor utilizado en la fase de testing. En cuanto a las tecnologías BigData, se utiliza *Spark* para el procesamiento de los datos geográficos, ya que es útil para la generación de grafos de contagios que son mostrados al especialista médico una vez calculados. Se ha utilizado Spark frente a otras opciones como *Hadoop* o *Flink* debido a su alta velocidad de procesamiento de datos en memoria.

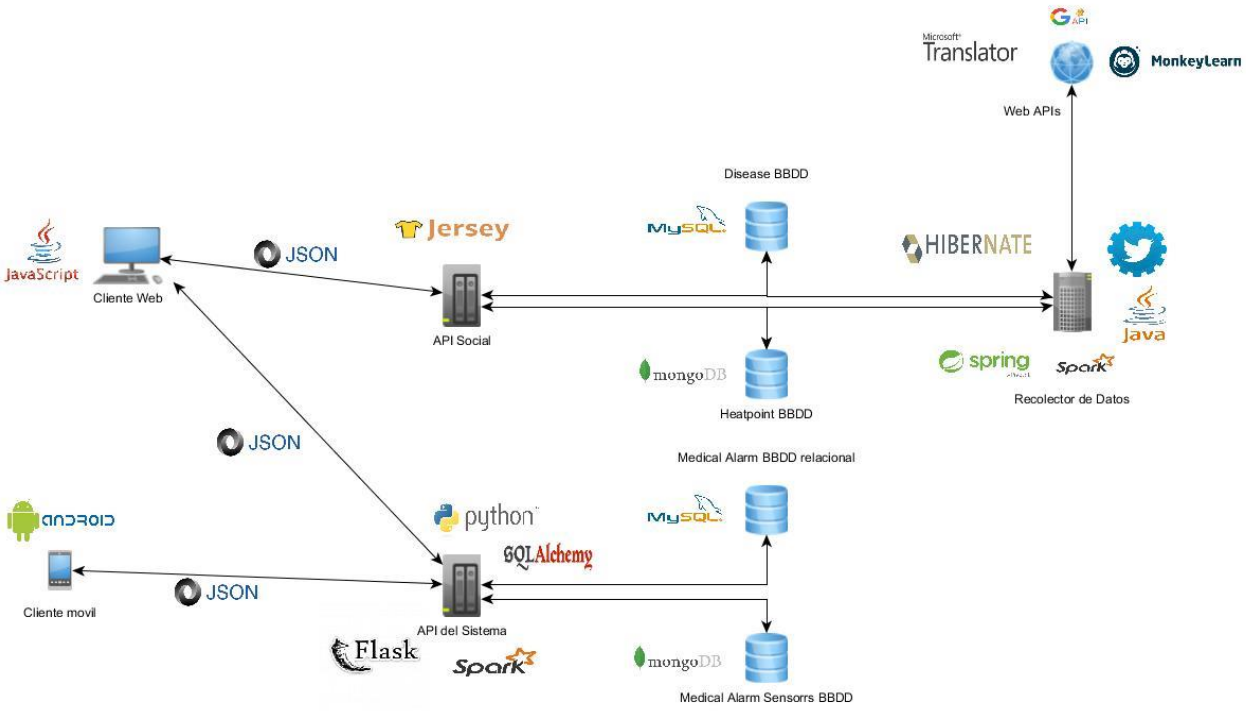
3.5 RESTful API

Por último, para el desarrollo de la API RESTful que se encarga de comunicar la aplicación móvil y la aplicación web con el servidor, se utiliza el framework *Flask* y como lenguaje de programación, *Python*. Existen varias alternativas como puede ser Django o Spring REST, pero se ha optado por Flask ya que para el desarrollo de este tipo de APIs es más adecuada y tiene una curva de aprendizaje menor.

Capítulo 4. Arquitectura de la aplicación

El sistema está formado por varios módulos, los cuales emplean tecnologías muy variadas. El sistema consta de un cliente web (para los médicos) y de una aplicación Android (para los usuarios), de cuatro bases de datos (dos relacionales y dos no relacionales) para almacenamiento de los datos, y por último de dos APIs RESTful, que permiten a la aplicación móvil y al cliente web realizar el cómputo de datos y la interacción con las bases de datos.

En el siguiente esquema se pueden ver todas las tecnologías empleadas en la arquitectura del proyecto, las cuales se explican en detalle justo a continuación.



34 - Figura 4.1 - Arquitectura

4.1 Aplicación Android

Aplicación para dispositivos Android, a partir de Android 4.1 por lo que será compatible con aproximadamente un 94% de los dispositivos Android del mercado. La compatibilidad *legacy* se tendrá en cuenta para futuras mejoras. Además, la aplicación ofrecerá la posibilidad de cambiar el idioma de la misma; inicialmente tomará el idioma predeterminado por el dispositivo móvil, pero se podrá elegir entre castellano e inglés. Más adelante se podrán ir implementando todos los idiomas que se necesiten.

Así pues, la aplicación es la fuente principal de datos, a través de los cuales se podrán inferir los contagios y focos. La labor principal de la aplicación es el envío de la localización y del momento o '*timestamp*' de los usuarios.

4.2 Aplicación web

El cliente web será el punto de interacción con los médicos o especialistas sanitarios. Mediante este cliente web se podrá establecer el punto de inicio de un contagio (es decir, el usuario) y la fecha, para que la ejecución del algoritmo alerte a los posibles afectados y permita la visualización de la evolución de una enfermedad, como el número de afectados, las zonas y datos estadísticos sobre estas. Además, se podrán buscar los posibles focos a partir de un listado de usuarios, para ver en qué punto pudo surgir un contagio.

4.3 Bases de datos

La base de datos se encontrará en el servidor y cumplirá la función de almacenar la información. Como ya se mencionó más arriba, el sistema está formado realmente por cuatro, dos relacionales y dos no relacionales.

Las dos primeras se utilizarán para el almacenamiento de los datos de usuario, de las enfermedades, contagios, notificaciones, etc., ya que son datos estructurados y que podrían ser modificados. Las dos últimas bases de datos serán no relacionales y almacenarán los datos no estructurados de localización y '*timestamp*' de los dispositivos. Estas BBDD serán no relacionales debido a la naturaleza no estructurada de los datos y la ausencia de una clave primaria. Además los datos se almacenarán y leerán, siendo necesarias solo las funciones de

lectura y escritura en *batch*, características que hacen más adecuada una BBDD no estructurada.

4.4 Servidor(es) y unidad(es) de procesamiento

El sistema constará de un servidor para el cliente web, donde también se encontrarán las BBDD. Además el sistema contará con un servidor destinado al procesamiento de los datos utilizando tecnologías Big Data.

4.5 RESTful API

Anteriormente hemos explicado los distintos componentes o módulos del sistema, pero por sí solos no pueden interactuar. Esta tarea se lleva a cabo mediante una API RESTful que permitirá tanto al cliente web como a la aplicación móvil enviar y recibir datos del servidor.

Así pues, una vez que se entienden los diferentes módulos que formarán el sistema, ya se puede entender de una manera más clara y detallada las tecnologías con las que han sido implementadas.

Capítulo 5. Modelo de datos

Una de las partes más importantes en un proyecto es el almacenamiento y tratamiento de los datos. Qué datos se guardan y cómo se guardan es vital para el desarrollo del trabajo. Por ello, en el proyecto, se dedica una parte notable del tiempo a la implementación de las bases de datos. Lo primero que se vio claro fue la existencia de cuatro bases de datos bien diferenciadas. El motivo fue que se contaba con dos tipos de datos bien diferenciados; por un lado, toda la información referente a los datos de los usuarios, contagios, médicos, enfermedades…y por otro, había que guardar todos los datos de los sensores de los móviles de los usuarios guardando gran cantidad de localizaciones de cada app descargada, y las localizaciones dadas por los datos sociales obtenidos de la web. Los dos tipos de datos eran muy diferentes al igual que su tratamiento, por lo que decidimos utilizar dos tipos de bases de datos.

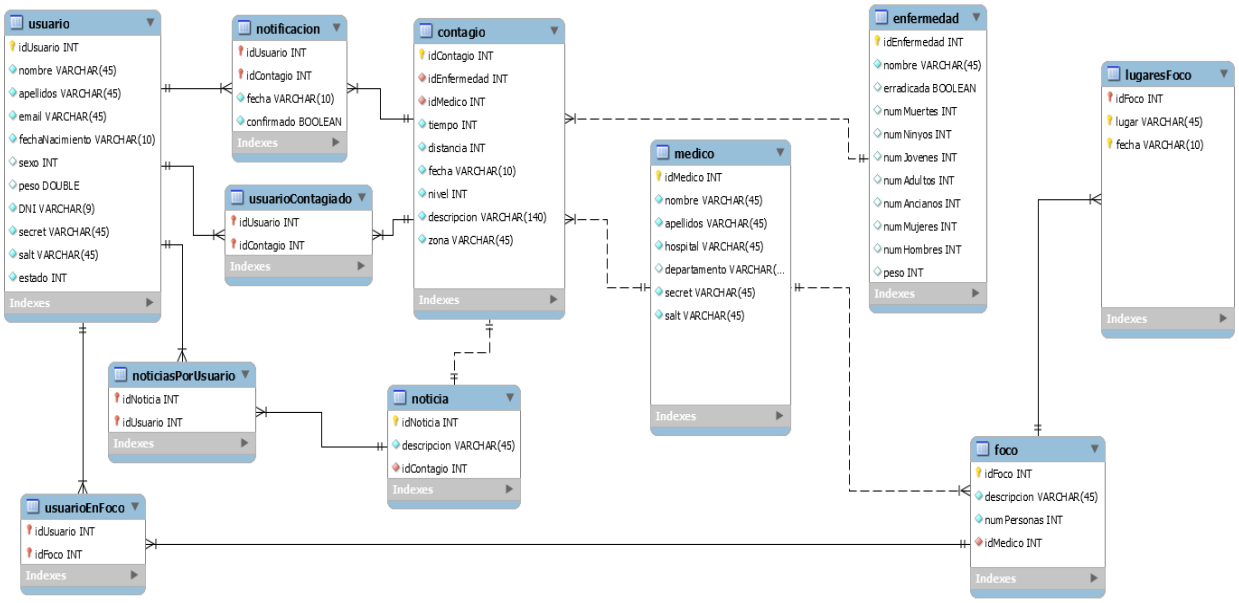
En primer lugar, se utilizó *MySQL* para guardar toda la información de usuarios, enfermedades y demás elementos mencionados anteriormente. *MySQL* ofrece un alto rendimiento y es estable. Se optó por una base de datos relacional porque la necesidad de actualizaciones, búsquedas concretas y borrados hacía que este tipo de bases fuera idónea para este caso.

En segundo lugar, había que enfrentarse a un almacenamiento de datos masivo, gran cantidad de información no estructurada y de carácter geoespacial; sin embargo, operaciones como actualizaciones o borrados no se contemplaban en esta sección. Todo esto condujo a utilizar una base de datos no relacional (NoSQL), optando por *MongoDB* ya que ofrece gran cantidad de funcionalidades relacionadas con geolocalización, grafos y operaciones matemáticas necesarias en nuestro proyecto.

Una vez conocidas las bases de datos utilizadas, sus motivos y los datos que van a almacenar, se describe con detalle cada una de ellas.

5.1 Base de datos relacional: MySQL

Para la implementación de esta base de datos se decide seguir el proceso aprendido a lo largo de la carrera por lo que se comienza realizando el diagrama entidad-relación, pasando al relacional y consiguiendo una primera aproximación de nuestra base de datos. Una vez obtenido esto y conociendo las tablas y relaciones existentes, gracias a la herramienta *MySQL Workbench* se implementa la base de datos de una manera gráfica y visual. Aquí se muestra el resultado:



35 - Figura 5.1.1 - Base de datos relacional

La base de datos cuenta con un total de 11 tablas relacionadas entre sí mediante claves foráneas y atributos. No cuenta con agregaciones o relacionales ternarias, se trata de una estructura sencilla y eficiente que se analiza a continuación:

Tabla **usuario**: contiene los atributos comunes a los usuarios y su clave primaria es un identificador auto incrementable.

- idUsuario: entero que forma la clave primaria.
- nombre: cadena que almacena sólo el nombre del usuario.
- apellidos: cadena que almacena los apellidos.
- email: cadena que almacena el correo electrónico del usuario. Podría haber sido clave primaria pero se pensó que podría repetirse en dos usuarios.

- fechaNacimiento: cadena que guarda la fecha de nacimiento (dd/mm/yyyy).
- sexo: entero que almacena 0 (hombre) o 1 (mujer). Puede ser null.
- peso: decimal que guarda el peso del usuario pudiendo ser null.
- DNI: cadena que almacena el DNI del usuario. Podría haber sido clave primaria ya que es único para cada usuario pero no se escogió por decisiones de diseño.
- secret: cadena que guarda la contraseña cifrada por motivos de seguridad.
- salt: cadena necesaria que almacena un código para obtener la contraseña.
- estado: guarda un entero (0, 1, 2, 3, 4) en función del estado (indefinido, sano, enfermo, curado, fallecido).

Tabla **medico:** contiene los atributos propios de un médico. Actualmente no se hace uso de esta tabla pues no se ha implementado ningún sistema de registro de médicos pero la base de datos lo soporta para posibles mejoras en el proyecto. La clave primaria será un identificador único auto incrementable.

- idMedico: entero que forma la clave primaria.
- nombre: cadena que almacena sólo el nombre del médico.
- apellidos: cadena que almacena los apellidos del médico.
- hospital: cadena que almacena el nombre del hospital al que pertenece el médico.
- departamento: cadena que guarda el departamento al que pertenece el médico para tener más detalle. Puede ser null por caber la posibilidad de no tener departamento.
- secret: cadena que guarda la contraseña cifrada por motivos de seguridad.
- salt: cadena necesaria que almacena un código para obtener la contraseña.

Tabla **enfermedad**: contiene un identificador y un gran número de atributos para almacenar los datos y estadísticas de cada enfermedad. Los contagios se van actualizando a medida que se detectan nuevos contagios pero no se eliminan ya que es un histórico de contagios necesario para la elaboración de estadísticas de los contagios provocados por esa enfermedad. Pueden ser null todos los campos ya que puede no haber ningún dato de esa enfermedad. Actualmente son fijas y la web no permite dar de alta enfermedades pero en un futuro no habría problema de incorporarlo puesto que está totalmente contemplado.

- idEnfermedad: entero que forma la clave primaria.
- nombre: cadena que almacena el nombre de la enfermedad.
- erradicada: booleano que guarda true o false en función de si está erradicada o no.
- numMuertes: entero que almacena el total de muertes provocadas por la enfermedad. cada vez que se confirma un fallecido se incrementa.

- numNinyos: entero que almacena el total de niños (0-12 años) contagiados.
- numJovenes: entero que almacena el total de jóvenes (13-25 años) contagiados.
- numAdultos: entero que almacena el total de adultos (26-65 años) contagiados.
- numAncianos: entero que almacena el total de ancianos (66 - años) contagiados.
- numMujeres: entero que almacena el total de mujeres contagiadas.
- numHombres: entero que almacena el total de hombres contagiados.
- peso: entero que guarda la media aritmética en kilogramos del peso de los usuarios contagiados.

Tabla **contagio:** contiene los atributos necesarios para conocer un contagio y una serie de referencias a otras tablas. Su clave primaria es un identificador auto incrementable.

- idContagio: entero que forma la clave primaria.
- idEnfermedad: referencia al identificador de la tabla enfermedad para saber de qué enfermedad proviene el contagio.
- idMedico: referencia al identificador de la tabla médico para saber el médico que ha declarado ese contagio.
- tiempo: entero que almacena en minutos el tiempo de exposición para que el algoritmo lleve a cabo la búsqueda de posibles usuarios contagiados.
- distancia: entero que almacena la distancia en metros introducida por el médico necesaria para el filtro de la búsqueda de posibles contagios.
- fecha: almacena en una cadena (dd/mm/yyyy) la fecha en la que se registró el contagio.
- nivel: entero que almacena el nivel de gravedad del contagio (1, 2, 3). Si un contagio se erradica se almacena temporalmente un 0 en vez de eliminarlo.
- descripción: pequeño texto a modo de descripción del contagio introducido por el médico.
- zona: cadena que almacena la zona (Madrid, Vallecas, Villalba…) del contagio.

Tabla **notificación**: necesaria para almacenar las notificaciones de posibles contagios detectados por el sistema enviadas a los usuarios. La clave primaria viene dada por el usuario (id) y el contagio (id).

- idUsuario: referencia al identificador del usuario que forma parte de la clave primaria.
- idContagio: referencia al identificador del contagio que forma parte de la clave primaria.
- fecha: cadena que almacena la fecha en la que fue enviada la notificación (dd/mm/yyyy).

- confirmado: booleano que almacena true o false en función de si el usuario ha acudido al médico y éste lo ha confirmado o de lo contrario está pendiente de la revisión.

Tabla **usuarioContagiado**: surge de la relación de varios a varios que existe entre usuarios y contagios puesto que un usuario puede estar en varios contagios y un contagio puede estar en varios usuarios. Al ser una relación n:m, surge una nueva tabla que contiene todos los usuarios que pertenecen a un contagio específico y su clave primaria se forma con la unión del identificador del usuario y el contagio.

- idUsuario: referencia al identificador del usuario que forma parte de la clave primaria.
- idContagio: referencia al identificador del contagio que forma parte de la clave primaria.

Tabla **noticia**: contiene toda la información relativa a noticias, las cuales son generadas automáticamente por la aplicación al detectarse un nuevo contagio o cuando se erradica alguno ya existente. La clave primaria la forma un identificador único auto incrementable y además existe una clave foránea referente al contagio del que se ha formado la noticia.

- idNoticia: entero que forma la clave primaria.
- descripcion: cadena autogenerada que contiene el texto informativo de la noticia.
- idContagio: entero y clave foránea que hace referencia al identificador del contagio al que pertenece la noticia.

Tabla **noticiasPorUsuario**: surge a partir de la relación existente entre usuario y noticia. Se trata de una relación de varios a varios, n:m. que da lugar a una nueva tabla. Es la encargada de almacenar todas las noticias que tiene un usuario y cuando éste las elimine de su móvil se borrarán de esta tabla. La clave primaria la forma el identificador de la noticia y el identificador del usuario.

- idNoticia: referencia al identificador de la noticia que forma parte de la clave primaria.
- idUsuario: referencia al identificador del usuario que forma parte de la clave primaria.

Tabla **foco**: contiene toda la información común a los focos entendiendo como foco un lugar concreto donde se ha podido originar una infección. Los focos pueden eliminarse y su borrado tiene lugar en cascada eliminándose todas las referencias de todas las tablas. La clave primaria es un identificador auto incrementable único y tiene una clave foránea que se asocia con el médico que ha buscado el posible foco a partir de un número determinado de pacientes.

- idFoco: entero que forma la clave primaria.
- descripcion: texto que contiene la descripción introducida por el médico acerca del foco.
- numPersonas: entero que indica el número de personas infectadas utilizadas para la búsqueda del foco.
- idMedico: clave foránea que hace referencia al identificador del médico.

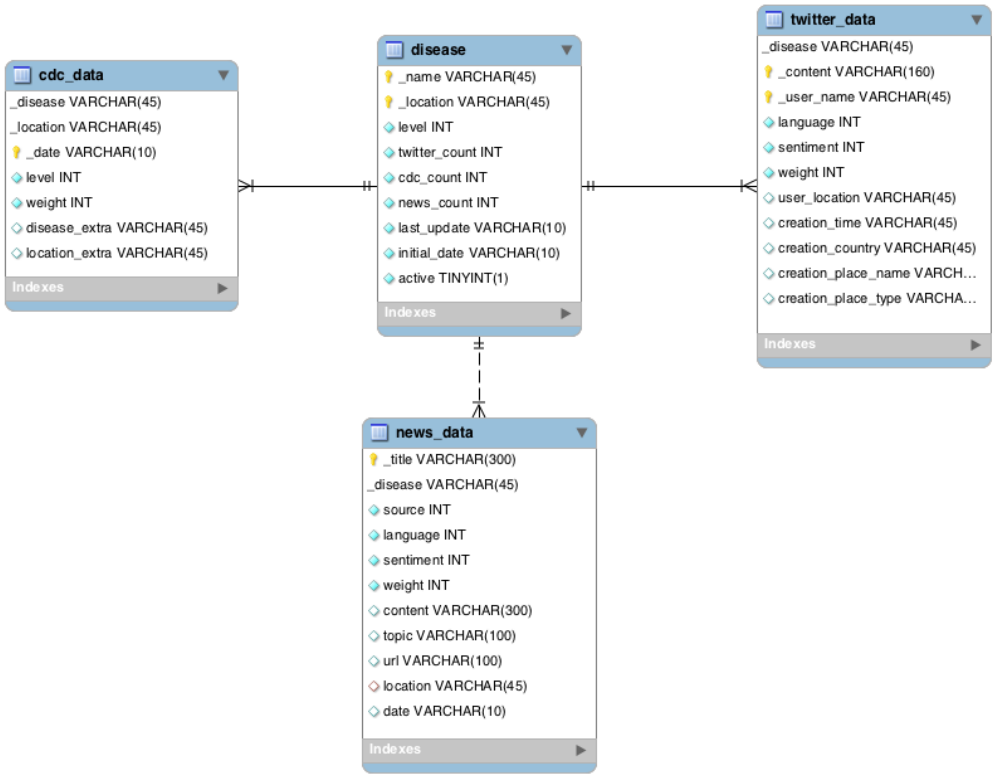
Tabla **usuarioEnFoco**: esta tabla se forma debido a la relación de varios a varios existente entre los usuarios y los focos ya que se puede dar que varios usuarios pertenezcan a un foco y que un foco esté en varios usuarios. En esta tabla se almacenan todos los usuarios que conforman un foco y la clave primaria está formada por los identificadores correspondientes al usuario y al foco.

- idUsuario: referencia al identificador del usuario que forma parte de la clave primaria.
- idFoco: referencia al identificador del foco que forma parte de la clave primaria.

Tabla **lugaresFoco**: esta tabla fue una incorporación final a la base de datos y en un principio no existía. El motivo de su incorporación fue la posibilidad de existir varios lugares como posibles focos de contagio. En un primer momento, al suponer que sólo se extraería un lugar exacto como foco, esta tabla no tenía sentido pero se cayó en la cuenta de que puede haber varios lugares como foco de contagio. Por esta razón es necesario almacenar todos los lugares que tiene un foco determinado. Todos los atributos de la tabla forman la clave primaria.

- idFoco: referencia al identificador del foco que forma parte de la clave primaria.
- lugar: cadena que almacena el nombre de la zona donde se ha detectado el foco y forma parte de la clave primaria.
- fecha: cadena que guarda la fecha (dd/mm/yyyy) en que se detectó el foco y forma parte de la clave primaria.

Además de la base de datos relacional mencionada anteriormente, el sistema cuenta con otra para la gestión de los datos sobre enfermedades provenientes de la web, es decir, de periódicos, twitter y la web oficial del CDC. El diagrama entidad-relación es el siguiente:



36 - Figura 5.1.2 - Base de datos relacional 2

Tabla **disease**: Esta tabla es el elemento principal de la base de datos, guardando en ella los datos correspondientes a una enfermedad en un lugar. Sus atributos son:

- _name: representa el nombre de la enfermedad y forma parte de la clave primaria.
- _location: representa la localización de la enfermedad y forma parte de la clave primaria
- level: representa el nivel de alerta de una enfermedad. Este puede ser 1,2 o 3.
- twitter_count: representa la cantidad de veces que ha sido mencionada la enfermedad en el lugar correspondiente en la red social Twitter.
- cdc_count: representa la cantidad de veces que ha sido mencionada la enfermedad en el lugar correspondiente en la página web oficial del CDC.
- news_count: representa la cantidad de veces que ha sido mencionada la enfermedad en el lugar correspondiente en los periódicos online.

- last_update: representa la última fecha en la que fue mencionada la enfermedad en el lugar correspondiente.
- initial_date: representa la fecha de creación del registro de enfermedad-localización.
- active: indica si la enfermedad está activa o no en la localización correspondiente.

Tabla **cdc_data**:

- _disease: representa la enfermedad a la cual se refiere la alerta del CDC y forma parte de la clave primaria.
- _location: representa la localización de la enfermedad a la cual se refiere la alerta del CDC.
- _date: indica la fecha en la cual se produjo la alerta y forma parte de la clave primaria.
- level: indica el nivel de alerta y forma parte de la clave primaria.
- weight: indica el peso asignado a la alerta.
- disease_extra: representa la especificación de una variante de una enfermedad común.
- location_extra: representa la especificación en la localización de la enfermedad.

Tabla **news_data**:

- _title: representa el título de la noticia y forma parte de la clave primaria.
- _disease: representa la enfermedad mencionada en la noticia y forma parte de la clave primaria.
- source: representa la fuente de la noticia. Es decir, el nombre del periódico.
- language: representa el idioma de la noticia (0 español, 1 inglés).
- sentiment: representa el sentimiento de la noticia (0 neutro, 1 positivo, -1 negativo).
- weight: representa el peso de la noticia.
- content: representa el contenido de la noticia.
- topic: representa el tema de la noticia.
- url: indica la dirección de la noticia.
- location: indica la localización mencionada en la noticia.
- date: indica la fecha de la noticia.

Tabla **twitter_data**:

- _disease: representa la enfermedad mencionada en el tweet y forma parte de la clave primaria.
- _content: representa el contenido del tweet y forma parte de la clave primaria.
- _user_name: representa el nombre del usuario que realizao el tweet y forma parte de la clave primaria.
- language: representa el idioma del tweet (0 español, 1 inglés).
- sentiment: representa el sentimiento del tweet (0 neutro, 1 positivo, -1 negativo).
- weight: representa el peso del tweet.
- user_location: representa la localización del usuario.
- creation_time: representa el momento de creación del tweet.
- creation_country: representa el país por defecto del usuario.
- creation_place_name: representa el nombre de la localización por defecto del usuario.
- creation_place_type: representa el tipo de localización por defecto del usuario.

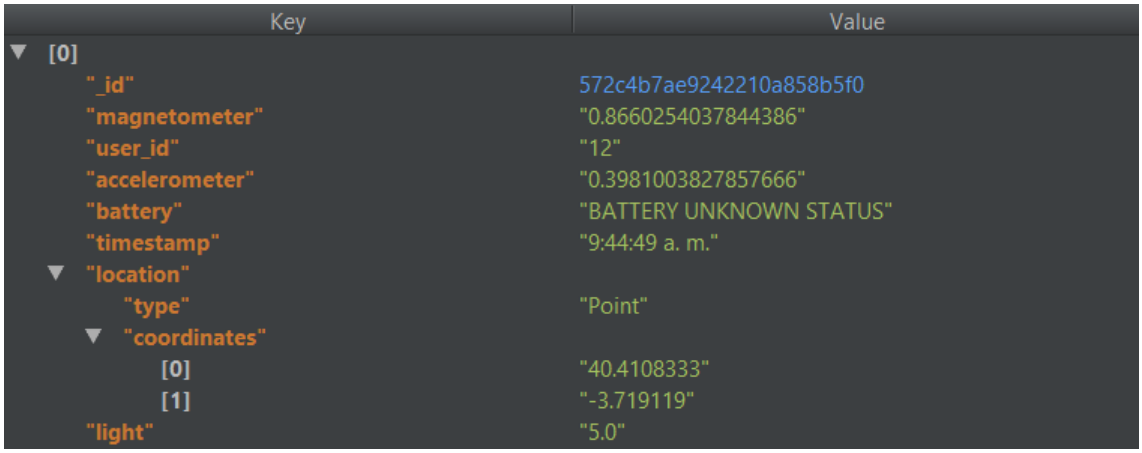
En los casos de las tablas para los registros de Twitter y los periódicos, solo el nombre de la enfermedad se usa como clave foránea de la tabla "*disease*" ya que puede darse el caso de que se mencione una enfermedad de forma genérica, y en este caso se entenderá que ser refiere a todas las localizaciones.

En el caso de la tabla para los registros del CDC ambos datos, nombre de enfermedad y localización, se utilizan como clave foránea ya que en este caso siempre viene dada la localización.

5.2 Base de datos no relacional (NoSQL): MongoDB

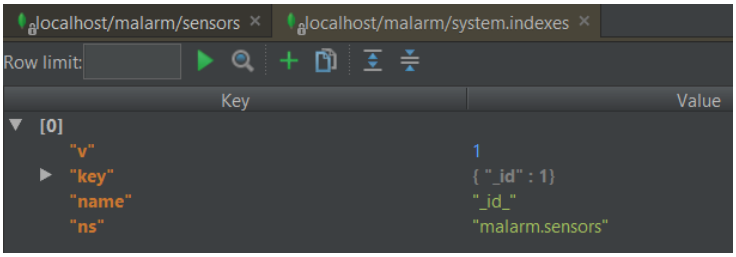
El uso se bases de datos no relacionales es muy cómodo ya que tiene un modelo de datos diferente. Está orientada a documentos, utilizando JSON como lenguaje de marcado, y carece de tablas y esquemas. Por esta razón, el diseño y la estructura de la base de datos son sencillos y sólo hay que centrarse en los campos que va a contener el documento. La base de datos de Mongo va a recibir cada cierto tiempo un conjunto de valores recogidos por la aplicación Android de cada usuario y va a ir ocupando una posición en el array del documento. En cada envío se almacenan los datos obtenidos de los sensores del teléfono (magnetómetro, acelerómetro, batería, luminosidad, la localización [latitud, longitud]), el timestamp para saber la hora exacta a la que se envió y muy importante, el identificador del usuario para poder

relacionar las localizaciones con los usuarios. Un ejemplo del JSON resultante sería el siguiente:



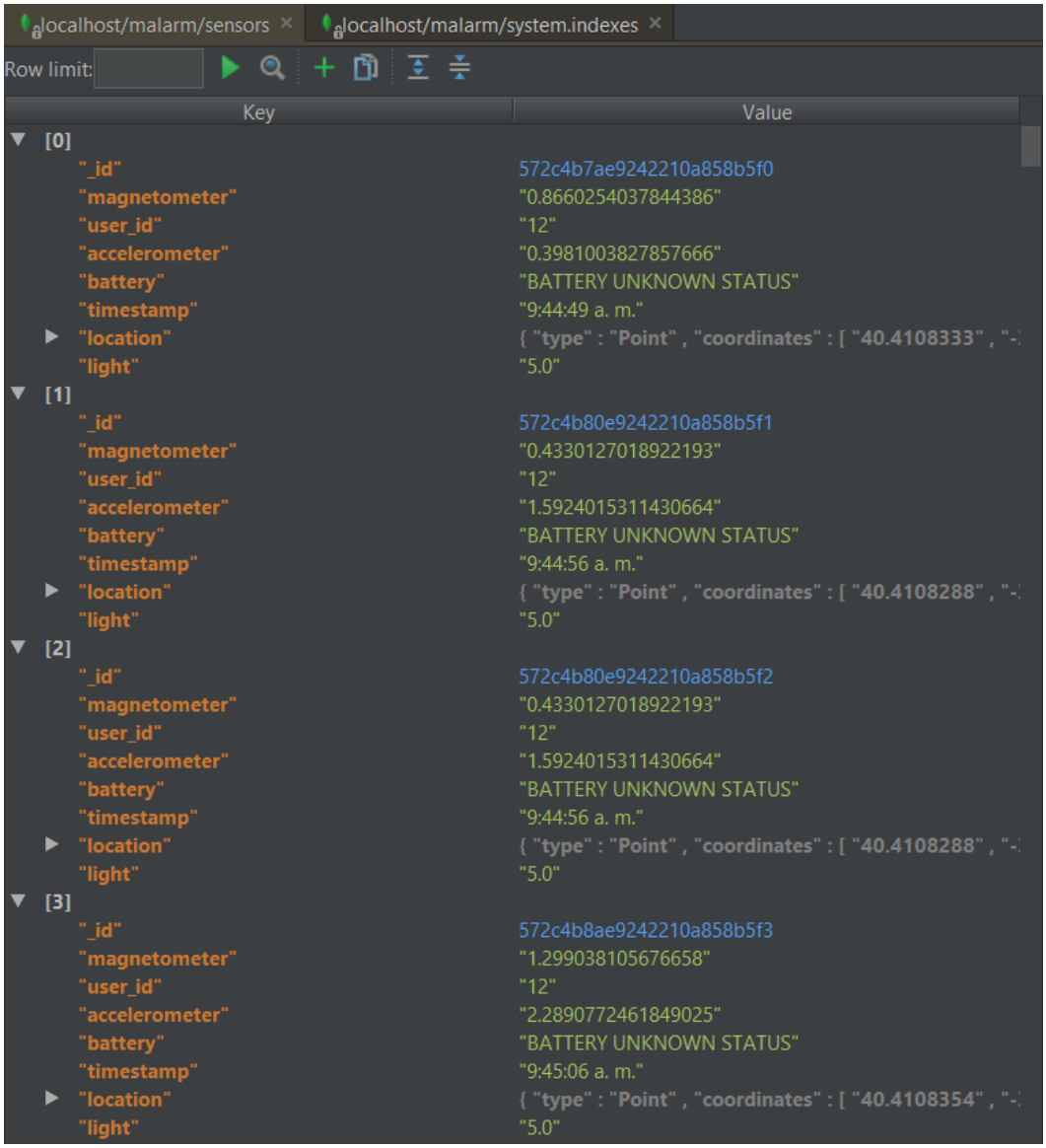
37 - Figura 5.2.1 - Base de datos Mongo 1

En el proyecto no se han utilizado los valores de los sensores del móvil exceptuando los del GPS pero se ha decidido guardarlos ya que la información puede resultar útil para futuras mejoras en el proyecto. Una idea que no se consiguió desarrollar del todo pero que está en proceso es la posibilidad de identificar si un usuario se encuentra en un espacio abierto o cerrado y para ello es necesario los valores de todos los sensores. Por este motivo, se ha decidido almacenar los datos aunque por el momento no se procesen. Con todo esto, se han guardado todas las localizaciones exactas en las que ha estado un usuario, su identificador y el momento exacto en el que ha estado. Gracias a todo esto y ejecutando los dos algoritmos de búsqueda de usuarios contagiados y localización del foco de una enfermedad se puede garantizar un resultado coherente y fiable. MongoDB ofrece además la posibilidad de indexar e insertar índices para que las búsquedas sean más rápidas y se puedan agrupar. Por eso, en la base de datos, se ha indexado por id para agrupar los usuarios y así conseguir que el tiempo en llevar a cabo las búsquedas disminuya considerablemente. Además se han utilizado índices de tipo "*2dsphere*" en el campo "*location*" para poder realizar operaciones geoespaciales sobre él.



38 - Figura 5.2.2 - Base de datos Mongo 1 (2)

Por último, destacar que no es necesario crear la base de datos o crear un patrón o estructura por defecto. Cuando la aplicación comienza a funcionar y MongoDB recibe el primer dato, se crea automáticamente la colección y se inserta el dato con la estructura que tiene. El resultado se puede ver a continuación:

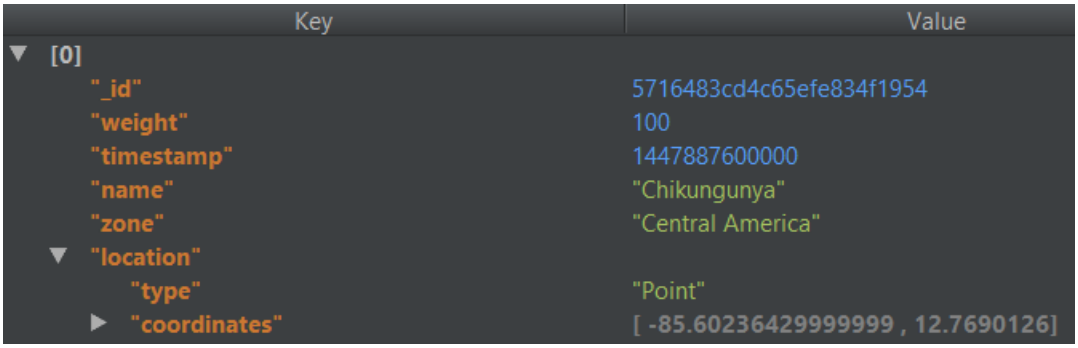


39 - Figura 5.2.3 - Base de datos Mongo 1 (3)

Este documento reflejado en la imagen anterior irá creciendo a medida que los usuarios vayan descargándose la aplicación y gracias a Mongo, la escalabilidad y el balanceado de carga no supondrán un problema para la sostenibilidad y el crecimiento del sistema.

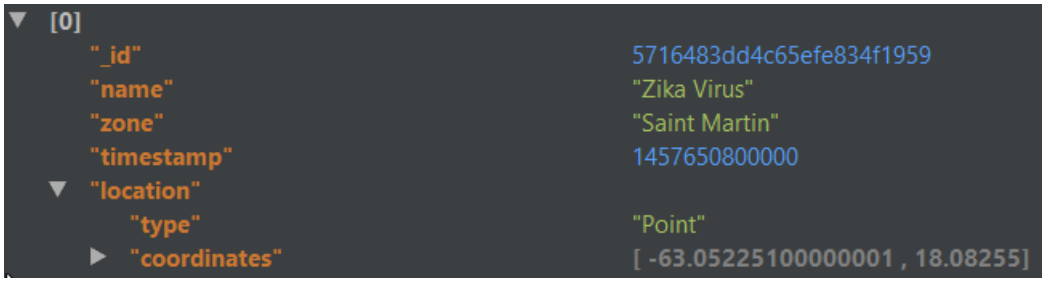
La base de datos llamada malarm contiene una colección llamada "*sensors*" donde realmente se guarda la información y donde van destinadas todas las consultas realizadas por la aplicación.

La segunda base de datos utilizada en *MongoDB* se utiliza para almacenar y consultar las localizaciones de los puntos geográficos en los cuales se ha mencionado una enfermedad en los datos sociales y para guardar el epicentro de estos; ambas colecciones utilizan índices de tipo "*2dsphere*" sobre el elemento "*location*". Para ello se utilizan dos colecciones con una estructura muy similar. La primera se llama "*heatpoint*" y almacena los puntos en los cuales ha sido mencionada junto con un peso para ser mas tarde representada en el mapa de calor.



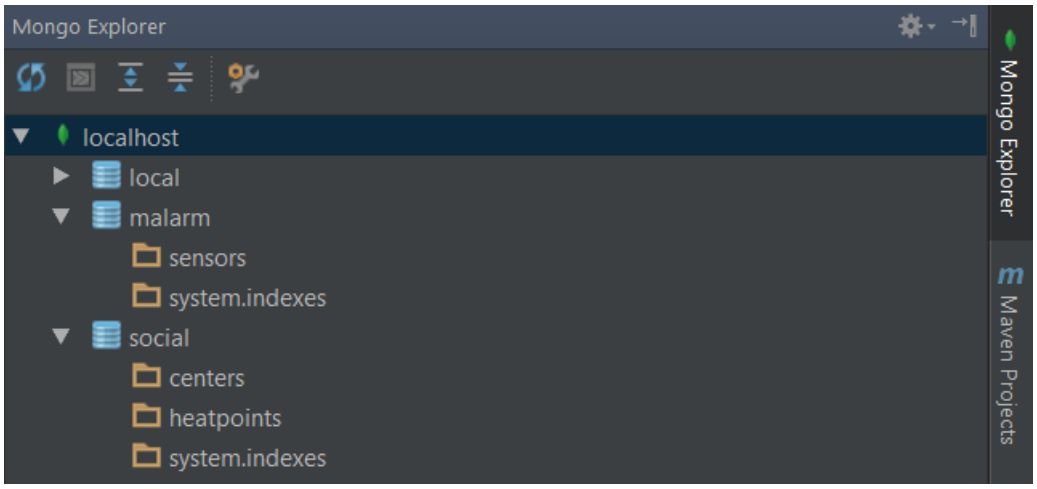
40 - Figura 5.2.4 - Base de datos Mongo 2 (1)

La segunda colección se llama "*center*" y representa los epicentros de las enfermedades. En este caso no se menciona un peso ya que sirve para tener un acceso más rápido y una diferenciación de que puntos representan el foco principal de la enfermedad.



41 - Figura 5.2.4 - Base de datos Mongo 2 (2)

Finalmente podemos ver las bases de datos y los esquemas. Destacar que las colecciones *"system.indexes"* son creadas automáticamente por *MongoDB* para almacenar los índices que se han ido creando en las bases de datos.



42 - Figura 5.2.6 - Esquema Mongo

Con todo esto quedan explicadas las cuatro bases de datos (2 MySQL y 2 MongoDB) utilizadas en el proyecto, sus propósitos y sus motivos argumentados. Las dos bases de datos se han conseguido integrar al sistema exitosamente gracias a plugins y librerías que han permitido las consultas y el uso de todo su potencial. Por medio de la API, encargada de determinar a qué base de datos hay que realizar cada consulta, el sistema es robusto, escalable e íntegro.

Capítulo 6. Diseño

Una pieza fundamental en el desarrollo de una aplicación o una página web es su diseño. Se suele imaginar a un informático escribiendo líneas de código a toda prisa, sin levantar la vista de la pantalla y sin prestar atención a nada más. Sin embargo, para desarrollar una buena aplicación hay mucho más trabajo que la implementación de la misma. Se conocen aplicaciones con muy buenos algoritmos y buena seguridad, pero con una interfaz descuidada y poco intuitiva que hacen que no sea usada, que no se extienda y que finalmente fracase. Antes de la codificación y la implementación de una aplicación hay un duro trabajo de síntesis, investigación y diseño que impulsan al proyecto y le ayudan a conseguir un producto completo y competitivo.

Una vez bien definida la idea del proyecto, cerrado el ámbito del sistema y concretada la existencia de los dos módulos de interacción con los usuarios, se comienza la etapa de diseño.

6.1 Diseño de la app MedicalAlarm

Lo primero a llevar a cabo es la identificación del escenario y de los usuarios de la app. El perfil de dichos usuarios es muy variado, ya que la app está pensada para que cualquier persona sin importar edad, experiencia o conocimiento pueda descargarla y empezar a utilizarla. Por este motivo, el perfil adecuado va desde un joven adolescente de 16 años, experto y acostumbrado al uso de aplicaciones, hasta una persona de avanzada edad, inexperta en el ámbito de la informática y con una curva de aprendizaje elevada. Teniendo en cuenta todo esto, se eligen una serie de principios de experiencia de usuario[3] para abarcar términos esenciales como:

- **Simplicidad**: relacionada directamente con la usabilidad, se busca ser mínimo, no cargar la interfaz de elementos y especialmente todos aquellos que estén presentes,

[3] filosofía de diseño que tiene por objetivo la creación de productos que resuelvan necesidades concretas de sus usuarios finales, consiguiendo la mayor satisfacción y mejor experiencia de uso posible con el mínimo esfuerzo.

dotarlos de una función bien definida que ayude al usuario y contribuya al cumplimiento del objetivo de la app.

- **Consistencia**: es importante respetar el aspecto visual de una app y del sistema operativo que tiene por debajo. Los usuarios de Android están habituados a ello y esperan que todas las apps se comporten de manera similar. Se trata de respetar los conocimientos y costumbres del usuario favoreciendo el uso intuitivo de la aplicación.

- **Navegación:** es un aspecto muy importante dentro de una aplicación, la forma en la que se navega entre contenidos y diferentes pantallas. Hay que evitar la sensación de desorientación del usuario y promover la navegación intuitiva siguiendo una serie de elementos estándar dentro de cada pantalla de la interfaz. Algunos elementos que Android utiliza para conseguir un uso fluido y sin esfuerzo son:

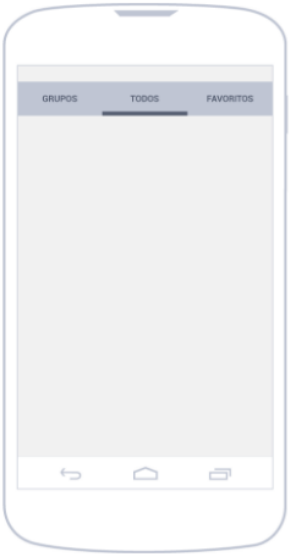
    - Pestañas: también llamadas *tabs*, se pueden utilizar para filtrar contenidos o cambiar entre pantallas que tienen la misma jerarquía, indicando siempre dónde se está y hacia dónde se puede ir. Una buena práctica es destacar siempre la pestaña seleccionada, mantener el orden y la ubicación inicial.

    - Listas: todo contenido dispuesto verticalmente y estructurado puede conformar una lista. Esta forma de mostrar ítems permite al usuario tocar alguno de ellos para obtener información complementaria.

    - Menús: elemento imprescindible en una app, existen varios tipos predominando en la actualidad el menú tipo cajón. Este patrón permite cambiar rápidamente de pantallas desplegándose lateralmente la pantalla seleccionada. Las ventajas del uso de este patrón son claras: mejor aprovechamiento del espacio, forma cómoda de navegar por el contenido y tener agrupadas las funcionalidades. El único inconveniente es la necesidad de desplegar el menú para poder visualizar todas las opciones.

    - Barra de acciones: en todos los sistemas, el compendio de acciones que se pueden realizar se representa por medio de iconos, por ello la correcta selección de estos recursos gráficos es fundamental. En Android, los botones de acción se encuentran en la zona superior derecha de la interfaz ordenados en función de la frecuencia de uso.

- Gestos: conocer todas las posibilidades que proporciona Android para navegar por la app a través de gestos con los dedos es muy importante para una experiencia única con la app. Incorporar gestos como *scroll*, *swipe* o pulsaciones cortas y largas facilitará la navegación y la fluidez.



43- Figura 6.1.1 - Prototipo pestañas

44- Figura 6.1.2 - Prototipo menú cajón

Además de todos estos patrones, se ha querido dar especial importancia a la usabilidad, fundamental en toda aplicación y para ello, se ha procurado cumplir los objetivos para la usabilidad de Nielsen[4]:

- **Facilidad de aprendizaje (Learnability):** cómo de fácil es para un usuario novato realizar una tarea en el sistema.
- **Eficiencia (Efficiency):** cómo de rápido puede realizar una tarea un usuario experto.
- **Memorabilidad (Memorability):** cómo de fácil es recordar cómo se usa un sistema tras haber pasado un tiempo sin usarlo.
- **Errores (Errors):** cuántos errores comete el usuario, cómo de graves son y cómo de fácil es recuperarse de ellos.

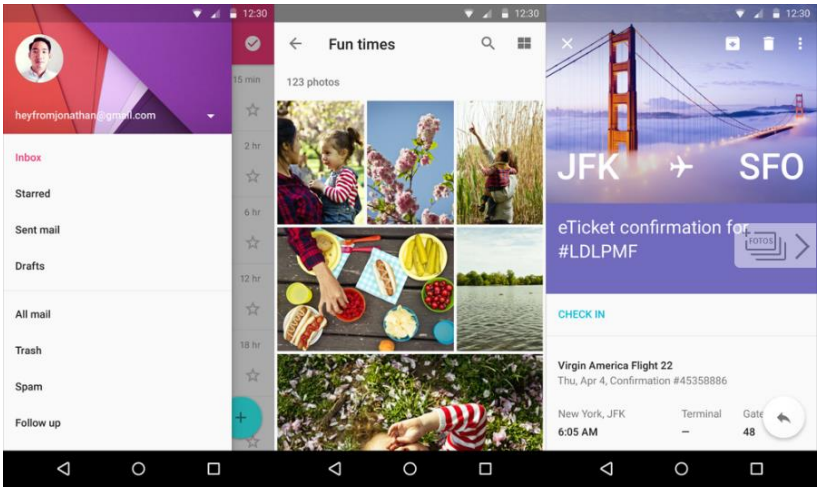
[4] Conjunto de 10 reglas o atributos de calidad que miden lo fáciles que son de usar las interfaces Web.

- **Satisfacción (Satisfaction):** cómo de agradable es usar el sistema desde el punto de vista del usuario.
- **Efectividad (Effectiveness):** cómo de bueno es el sistema haciendo lo que se supone para lo que está diseñado.

Además de tener en cuenta todos estos patrones y modelos de diseño, se ha seguido la guía *Material Design* que ofrece Google para llevar a cabo una buena interfaz cumpliendo sus reglas. Así pues, la aplicación ha sido diseñada basándose en Android Lollipop 5.0 introduciendo nuevos principios relacionados con las capas, las texturas y las formas.

¿Qué es Material Design?

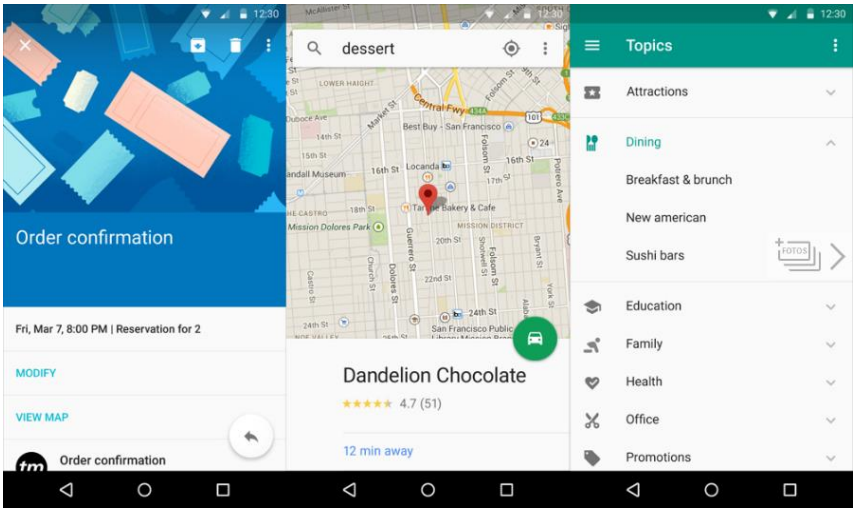
Material Design es un concepto, una filosofía, unas pautas enfocadas al diseño utilizado en Android, pero también en la web y en cualquier plataforma, ya que sigue los patrones *responsive*.



45 - Figura 6.1.3 - Material Design

Recibe su nombre por estar basado en objetos materiales, piezas colocadas en un espacio y con un tiempo determinado. Es un diseño donde la profundidad, las superficies, los bordes, las sombras y los colores juegan un papel principal. Busca aproximarse a la realidad, algo que en un mundo donde todo es táctil y virtual es difícil. Material Design quiere guiarse por las leyes de la física, donde las animaciones sean lógicas, los objetos se superpongan pero

no puedan atravesarse el uno al otro. Estas técnicas se aplican en Android delimitando claramente el tipo de menús, los botones y los tipos de imágenes a elegir. Google recomienda usar paletas de colores primarios que creen contraste para atraer la atención del usuario integrándolos en uno de los temas proporcionados, Holo Light u Holo Dark. Además es conveniente usar la tipografía de Android, Roboto. Por último, otro elemento clave es la luz y las sombras. Una iluminación realista proporciona indicios de cómo se comportará un elemento y en qué nivel se encuentra. El uso adecuado del movimiento, las sombras y las formas al igual que vibraciones o gestos intuitivos nos acerca a una de las partes más importantes que han influido en el éxito de Android que es el feedback[5], el cual nos ayuda a interactuar en todo momento con el usuario.



46 - Figura 6.1.4 - Material Design 2

Una vez comprendidos, estudiados y analizados todos y cada uno de los patrones citados y las técnicas recomendadas por Google y su *Material Design*, comenzó el diseño de la aplicación y para ello se siguieron una serie de etapas:

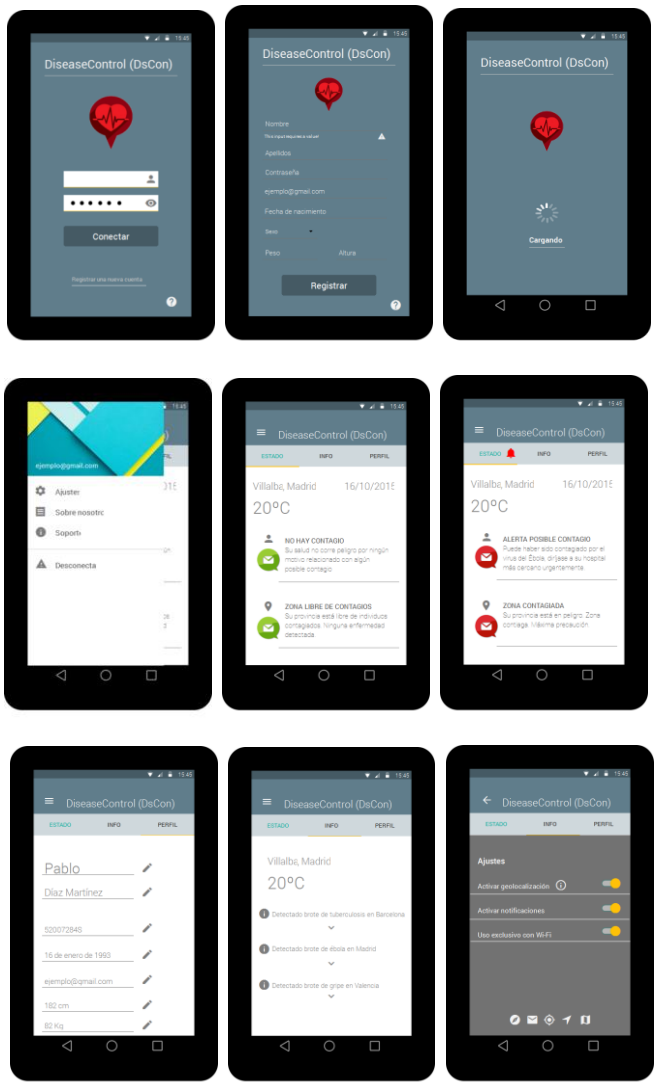
- En la primera etapa, se llevaron a cabo unas primeras aproximaciones sobre papel mediante una serie de bocetos con un grado de detalle muy bajo.
- En la segunda etapa, con las ideas claras de la etapa anterior, las pantallas definidas y la mayoría de las interacciones, se pasó a realizar una serie de bocetos con un detalle mayor por medio de una herramienta, un software de diseño y maquetación muy

[5] Término utilizado para denominar el mecanismo por el cual una cierta proporción de la salida de un sistema se redirige a la entrada, con objeto de controlar su comportamiento

completo para aplicaciones móviles y web llamado JustInMind. Una vez acabados los diseños, se hizo una primera valoración realizando una serie de entrevistas a familiares, dejándoles interactuar con la interfaz mediante varias demos. Gracias a estas entrevistas, se tomaron notas sobre cambios a realizar para mejorar la interfaz y acercar más el producto al usuario.

- En la tercera y última etapa, se retocaron y modificaron las interfaces obtenidas en la segunda etapa, incorporando más detalle y acercándose a la realidad. Al obtener estos bocetos con alto nivel de detalle, la posterior implementación de la interfaz no resultó tan tediosa y complicada.

A continuación se muestran los resultados obtenidos:



47 - Figura 6.1.5 - Prototipos finales

El primer paso fue incorporar un menú desplegable (*Navigation Drawer*) de tipo cajón donde agrupar todas las opciones ofrecidas al usuario. Para llegar a este menú se puede pulsar el botón en la esquina superior izquierda, o mediante un gesto con el dedo de deslizar en el que el menú aparece desde el lado izquierdo hacia el derecho. El resto de la aplicación se diseñó mediante un sistema de pestañas o tabs. Existen tres pestañas diferenciadas con distinta información en cada una de ellas pudiendo intercambiarlas pulsando en cada pestaña o realizando un gesto de swipe con el dedo. Toda la aplicación gira en torno a estas tres pestañas con distintos botones y listas en cada una de ellas, siendo cada una independiente y evitando la confusión del usuario. Por otra parte, en la pantalla de ajustes a la cual se accede mediante el menú desplegable, se ofrece la posibilidad de configurar varios aspectos como el uso de Internet, el idioma de la app o la aparición de notificaciones.

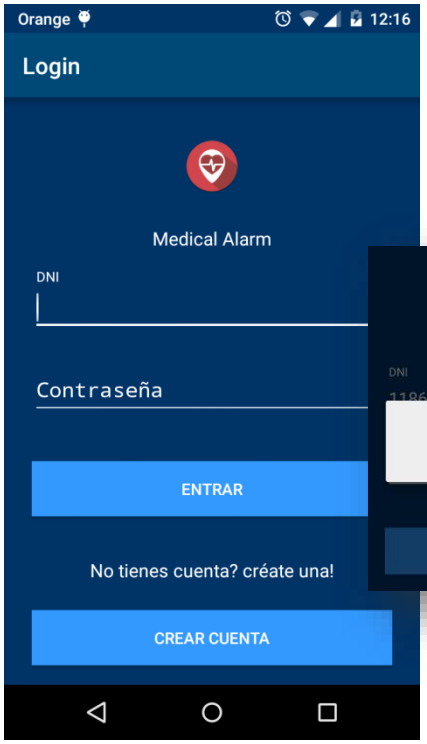
Debido a la relación con la sanidad, se optó por escoger una gama de azules. Finalmente se escogió como primario el *BLUE* y como secundario el *CYAN* y a partir de eso se estableció la paleta de colores de la aplicación.

48 - Figura 6.1.6 - Paleta colores

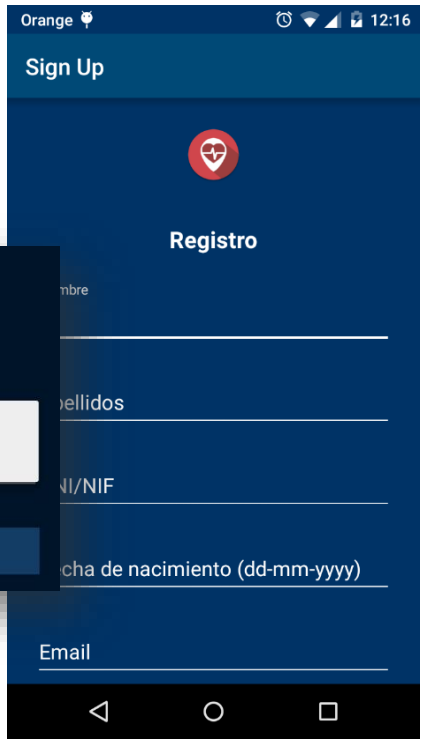
Por último, quizá no tan relevante para el proyecto, pero sí para el futuro del sistema, se decidió buscar un logo y un nombre al mismo. Así pues, se elaboró un logo plano, siguiendo las texturas de *Material Design*, buscando unir iconografía relacionada con la sanidad (un corazón) y la geolocalización (una marca de un lugar del GPS). Se denominó Medical Alarm, siendo éste el resultado:

49 - Figura 6.1.7 - Logo

Una vez realizados los prototipos de la aplicación móvil y gracias al software *JustInMind*, se obtuvo un método de exportación a página web con el cual se pudo realizar una serie de pruebas con personas y ver la experiencia de usuario. Se escogieron familiares y amigos, y siempre con dos integrantes del grupo presentes, se fue haciendo un seguimiento de la experiencia de ese usuario y apuntando las respuestas a una serie de preguntas guiadas que cada persona iba respondiendo. Una vez obtenidos los resultados y analizados, se realizó algún pequeño cambio y se dio paso a la implementación de la aplicación final. En este sentido, se seguieron los prototipos y únicamente se tuvieron que cambiar pequeños detalles e incorporar algunos elementos más, que sólo cuando empezó la implementación surgieron. Los resultados de la interfaz definitiva fueron los siguientes:



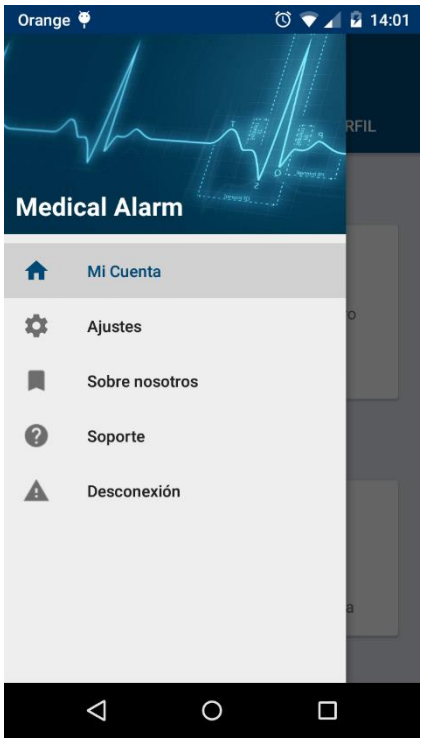
51 - Figura 6.1.8 – Login



50 - Figura 6.1.9 - Registro

Justo encima se muestran las pantallas de login y registro, las primeras que el usuario visualizará cuando arranque su aplicación. Los campos cuentan con sus comprobaciones y alertas correspondientes y una vez que se han introducido todos los datos correctamente, se mostrará un pop-up, indicando que se está accediendo a la pantalla principal, elemento mediante el cual el usuario es consciente de que los datos introducidos son correctos y que debe esperar para visualizar todos los datos en la pantalla principal.

53 - Figura 6.1.10 - Inicio

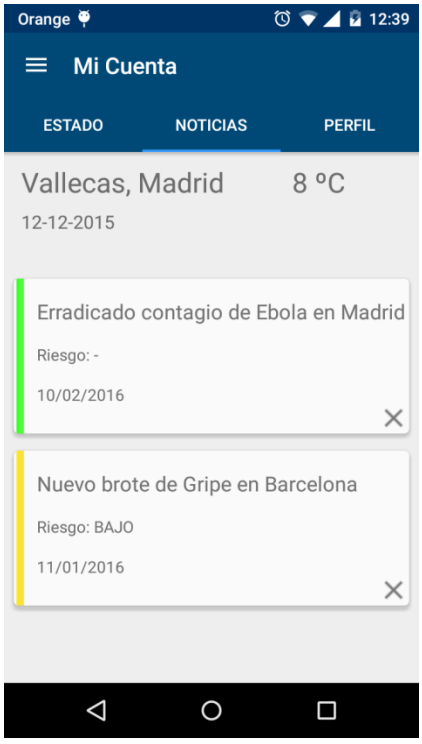


52 - Figura 6.1.11 - Menú

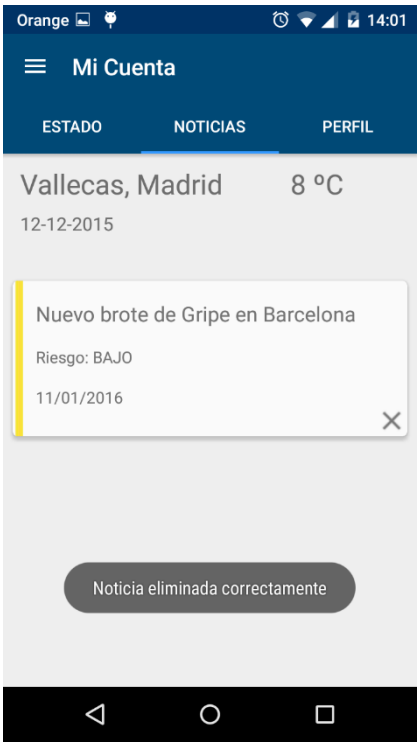
Se trata de la vista perteneciente a la primera pestaña, la cual es cargada y mostrada por defecto, convirtiéndose en la pantalla principal de la aplicación. Se destaca el uso de los principios de diseño y de elementos Material Design. En esta pantalla se puede observar la existencia de un sistema de pestañas en el que se resalta la pestaña actual como señal visual para el usuario, para que sepa dónde está y las posibles acciones que puede llevar a cabo. Se permiten los gestos como *swipe* para pasar de una pestaña a otra o simplemente pulsar la pestaña elegida; además, mediante un *swipe* desde el lateral izquierdo de la pantalla se desplegará un menú de cajón, característico en *Material Design* y en las aplicaciones actuales *Lollipop Android 5.0*. Las imágenes son planas, jugando con las texturas y las cards dando profundidad y resaltando los mensajes importantes al usuario. Con esto se potencia la memorabilidad y la facilidad de aprendizaje. También se añadió otro gesto de recarga para actualizar la información siempre, proporcionando feedback al usuario para que sepa qué está sucediendo en todo momento y tenga el control de la aplicación.

54 - Figura 6.1.12 - Feedback

Las otras dos vistas correspondientes a las pestañas restantes siguen ofreciendo una serie de características propias de este paradigma de diseño, con el control de los espacios en blanco, las sombras y los elementos gráficos que favorecen la conformidad del usuario.



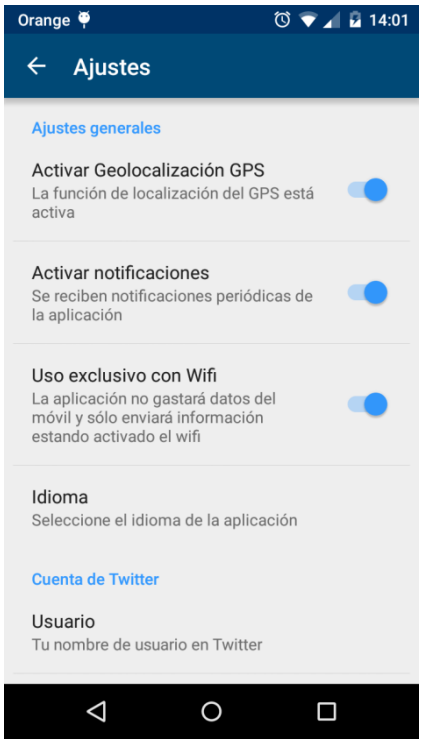
55 - Figura 6.1.13 - Vista Noticias



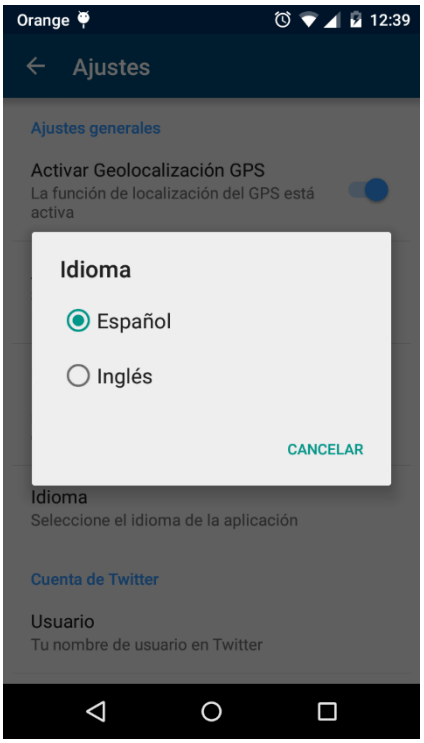
56 - Figura 6.1.14 - Eliminar noticia

Al pulsar la X para eliminar una noticia, se muestra un mensaje indicando que la noticia se ha eliminado al mismo tiempo que desaparece de la vista.

Una vez expuestas, y explicadas las vistas principales de la aplicación, queda por mostrar la vista de ajustes, la cual ofrece una serie de configuraciones y a la que se accede por medio del menú cajón disponible, realizando *swipe* desde el lateral izquierdo de la pantalla hacia la derecha. Todos los ajustes proporcionan feedback al usuario, despliegan diálogos de conformación o de atención, haciendo la experiencia del usuario más gratificante.



57 - Figura 6.1.15 - Vista Ajustes



58 - Figura 6.1.16 - Ajuste idioma

Con esto quedan mostradas todas las vistas de la aplicación Android, y explicadas y justificadas todas las decisiones de diseño. Conviene resaltar que en este apartado de la memoria se explica el proceso de diseño de la aplicación, haciendo un recorrido por la misma desde el punto de vista del diseño, la maquetación y la disposición de los elementos.

Realizado el diseño de la app, quedaba por diseñar la web antes de poder pasar al desarrollo.

6.2 Diseño de la página web

Para llevar a cabo el diseño de la web, se decidió usar Bootstrap, un framework o conjunto de herramientas de código abierto para diseño de aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basados en *HTML* y *CSS*, así como extensiones de *Javascript* opcionales adicionales. Está disponible en *GitHub*. Es modular con un sistema de cuadrilla y responsive (diseño sensible) y gran cantidad de plug-ins; además es compatible con la mayoría de navegadores. Es una herramienta activa, en constante evolución y cada vez con más peso en el mundo de las aplicaciones web.

A pesar de las funcionalidades que ofrece Bootstrap, fue necesario pensar qué diseño iba a tener la web, teniendo en cuenta los usuarios de la misma y siguiendo una serie de principios que hicieran de la web una herramienta óptima y útil para sus usuarios. Para definir todos estos elementos, se siguieron una serie de pasos al igual que con la aplicación móvil, empezando por la identificación y el análisis de los usuarios de la aplicación.

Los usuarios de la página web iban a ser médicos profesionales dentro de un hospital o centro médico, por lo que se trataba de gente cualificada, con un lenguaje técnico y preciso. Sin embargo, el manejo de sistemas informáticos no tenía por qué ser elevado por lo que había que llevar a cabo una web amistosa, consistente y fácil de utilizar. Era importante tener presente en todo momento que los destinatarios eran médicos, exclusivamente médicos, por lo que cada detalle de la página debía adecuarse al ámbito de la medicina. Una vez identificados y acotados los usuarios, era momento de comenzar el diseño y, para ello y con ayuda de Bootstrap, se siguieron una serie de principios de diseño para ganar calidad en el proyecto.

Existen gran cantidad de guías prácticas y principios de diseño, pero en el proyecto se han escogido una serie de principios que se adecúan mejor al proyecto:

- **Principio de proximidad**: la proximidad se puede explotar de dos formas, agrupando los controles relacionados o agrupar los controles que representan secuencias de acciones. Tiene una serie de ventajas, como saber que al agrupar elementos similares entre sí conseguimos que el usuario sepa que están relacionados o relacionarlos mentalmente y recordarlos como grupo (chunk).

  - Ley de Fitt: muy relacionada con el principio de proximidad está la Ley de Fitt, la cual enuncia que el coste en tiempo para activar un control es proporcional a la distancia a la que se encuentra e inversamente proporcional al tamaño. Pueden ser consecuencias de dicha ley que los movimientos sean cortos y precisos o largos e imprecisos, que el desplazarse a un elemento pequeño ubicado a mucha distancia es costoso y que los bordes y las esquinas son más fáciles de apuntar.

- **Principio de consistencia**: los usuarios aprenden más fácilmente los conceptos que son consistentes con su conocimiento previo. Todos los controles con funcionalidad similar deberían tener una apariencia y exhibir un comportamiento similar. Por otra parte, el principio aconseja desarrollar aplicaciones consistentes con las expresiones habituales dentro de un sistema específico, como emplear los mismos atajos de teclado, los mismos elementos visuales o los mismos procedimientos. Llevar a cabo y seguir este principio es importante para la memorabilidad y la facilidad de aprendizaje.

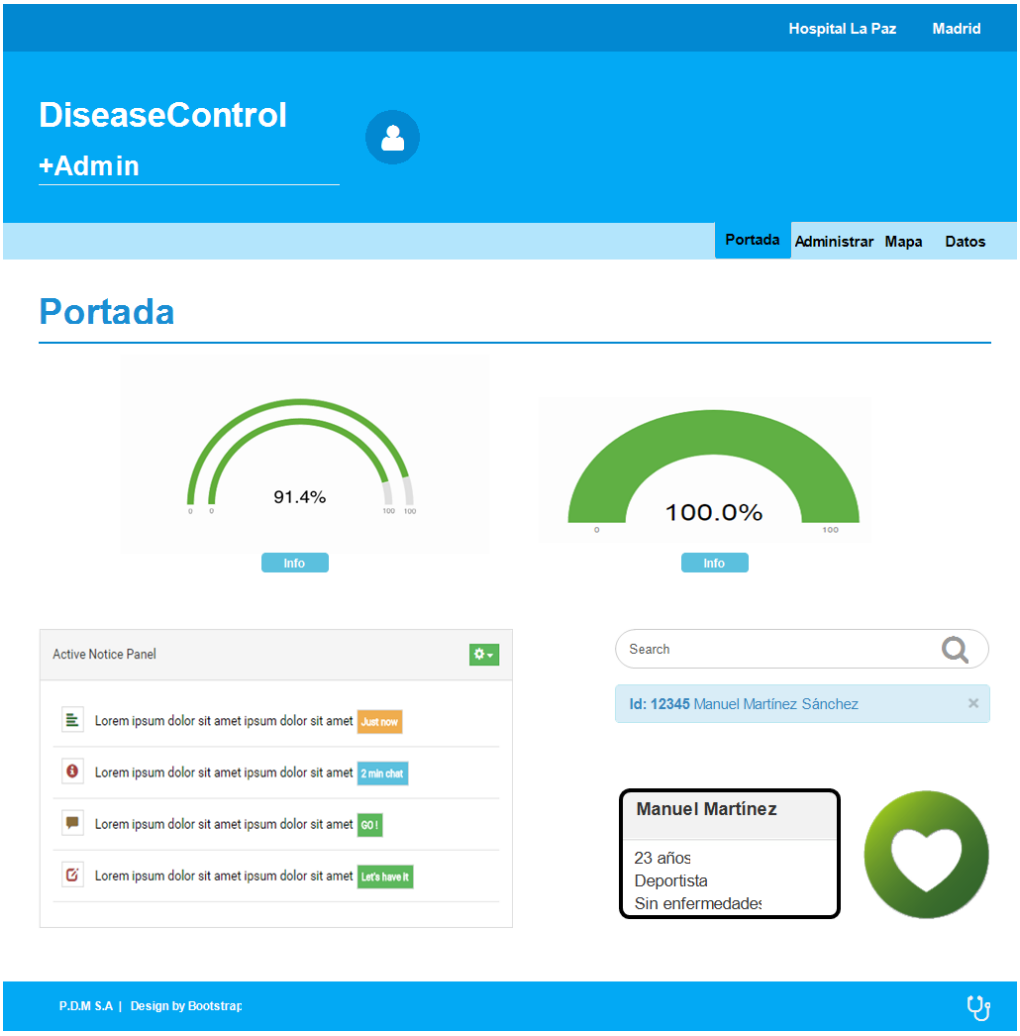
- **Principio de visibilidad y feedback**: tiene que ver con cómo la interfaz emplea distintos mecanismos para transmitir su estado actual y las acciones posibles (o recomendables) en un momento determinado. Hay mecanismos para dirigir la atención como la tipografía, la opacidad, el tono o el relieve. Además, el feedback juega un papel relevante y a veces no se le da importancia. El sistema siempre debe reaccionar ante cualquier acción, especialmente si la respuesta puede tardar debido a un retardo en el sistema, o el sistema de input de acciones no es fiable. Se deben facilitar barras de progreso cancelables y feedback sobre la duración esperada, y permitir al usuario saber qué acciones están disponibles.

- **Principio de gestión del estado visible**: evitar las brechas entre el estado de la aplicación y el estado mental del usuario mostrando el estado de navegación, el estado del modelo y el estado de la interfaz.
  - Estado de la navegación: informa al usuario de dónde se encuentra en un flujo de ejecución, como por ejemplo migas de pan en sitios web o pestañas resaltadas.
  - Estado del modelo: la interfaz debe sugerir el estado interno en el que se encuentra la aplicación, mostrando las propiedades del objeto seleccionado, el estado general del sistema o el estado de la última operación.
  - Estado de la interfaz: la propia interfaz debe ofrecer información sobre el estado actual.
- **Principio de libertad y control del usuario**: el usuario debe sentirse con el control del proceso. Debe sentir que puede explorar la interfaz cómodamente y que es él quien

controla al programa y no al revés. Requiere encontrar un equilibrio entre llevar al usuario a donde queramos, y permitirle ir a donde quiera. Esto se consigue ofreciendo al usuario opciones y no preguntas, ofrecerle las herramientas que necesita o reducir la fricción cognitiva tratando de anticipar los modelos mentales del usuario.

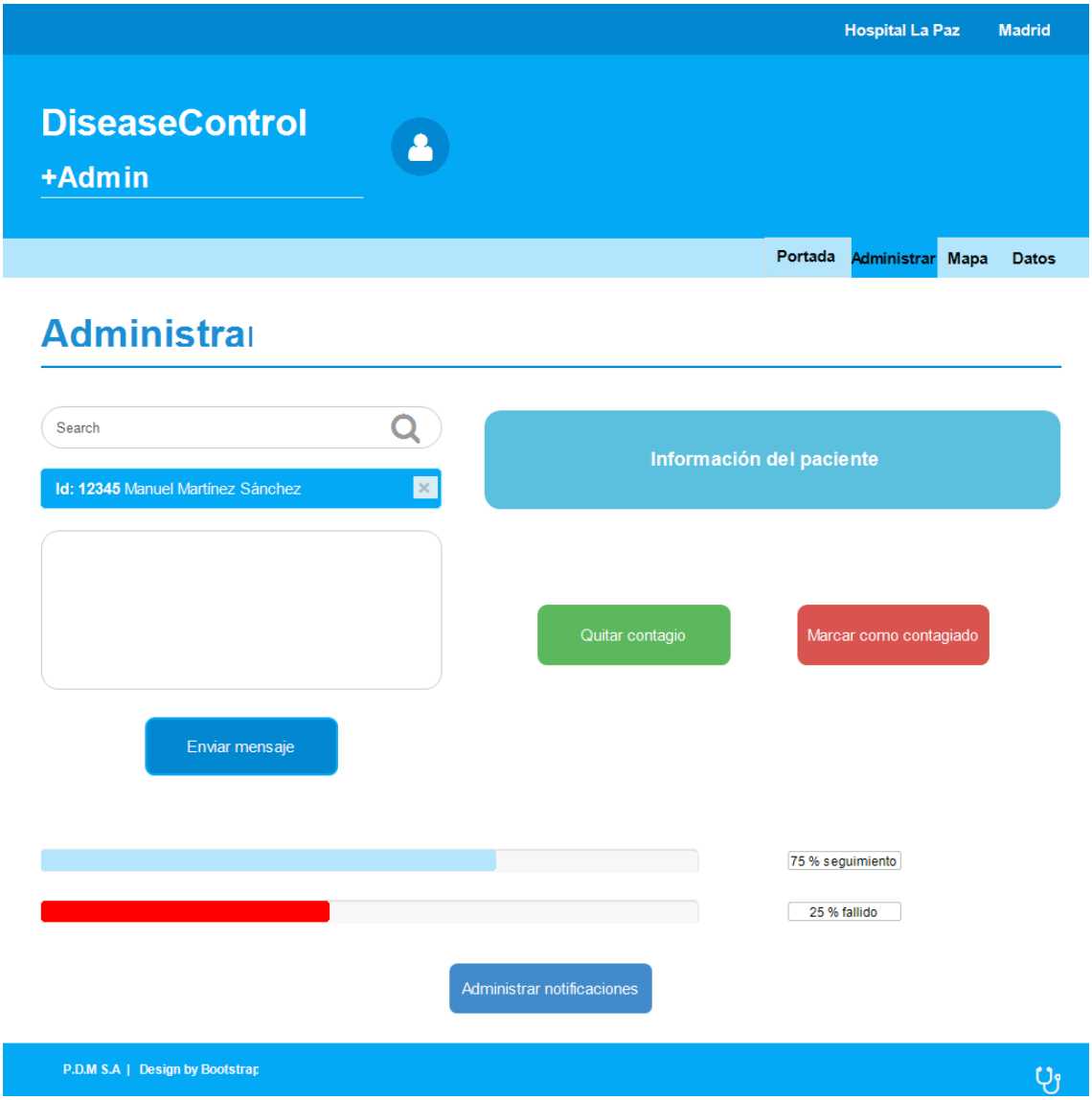
Teniendo todos estos principios en cuenta, analizando la disposición de los elementos de la web y cada interacción, se realizaron los siguientes prototipos:

La portada cuenta con unos gráficos genéricos del estado del sistema, un buscador de pacientes y un panel principal.



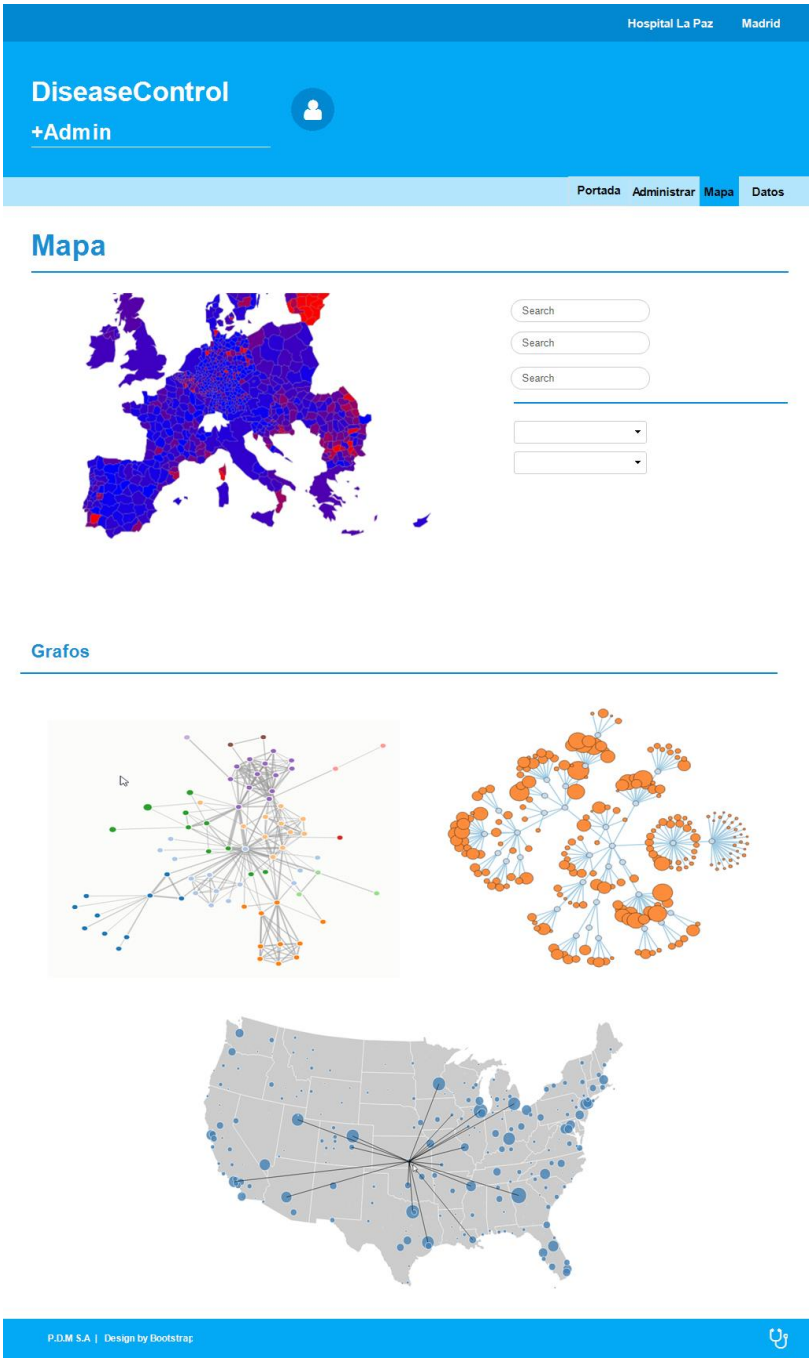
59 - Figura 6.2.1 – Prototipo Portada

La siguiente pestaña corresponde a la de administración, donde se lleva el control del estado de cada paciente con un buscador y un panel para su configuración y, por otro lado, las notificaciones enviadas.



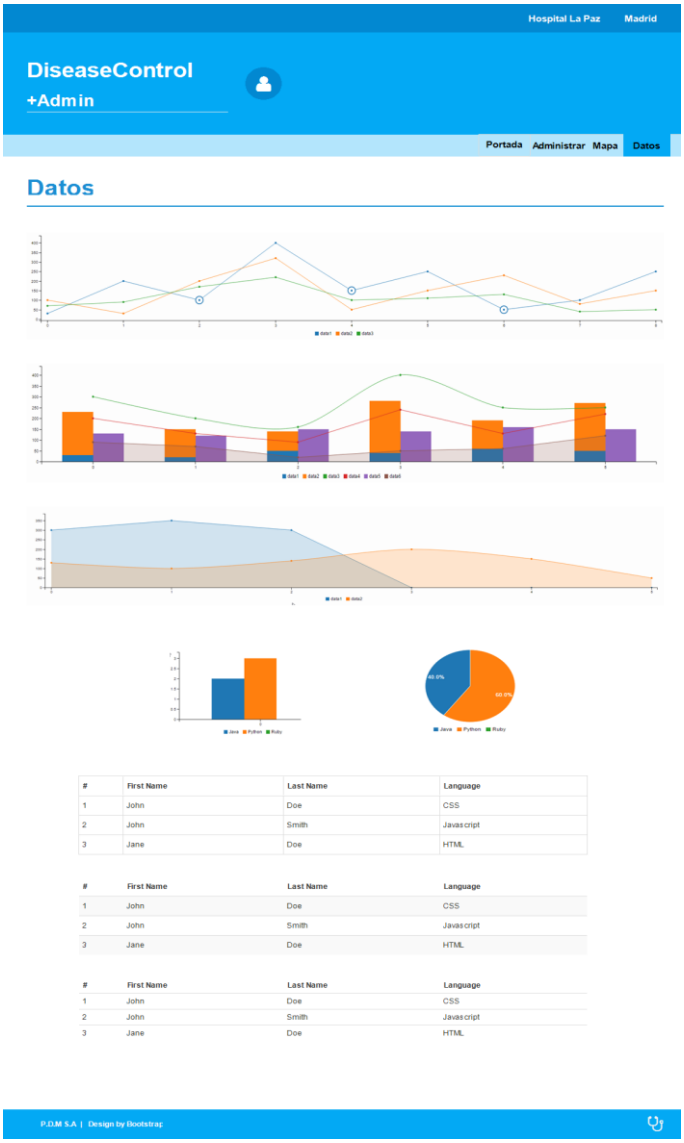
60 - Figura 6.2.2 – Prototipo Administrar

El elemento principal de la siguiente pestaña, como su nombre indica, es el mapa de calor de España con un punto de calor para cada contagio activo, siendo una ayuda visual para el médico en todo momento. Además, se detallan una serie de grafos donde se pueden apreciar las relaciones entre los contagios y los diferentes focos.



61 - Figura 6.2.3 – Prototipo Mapa

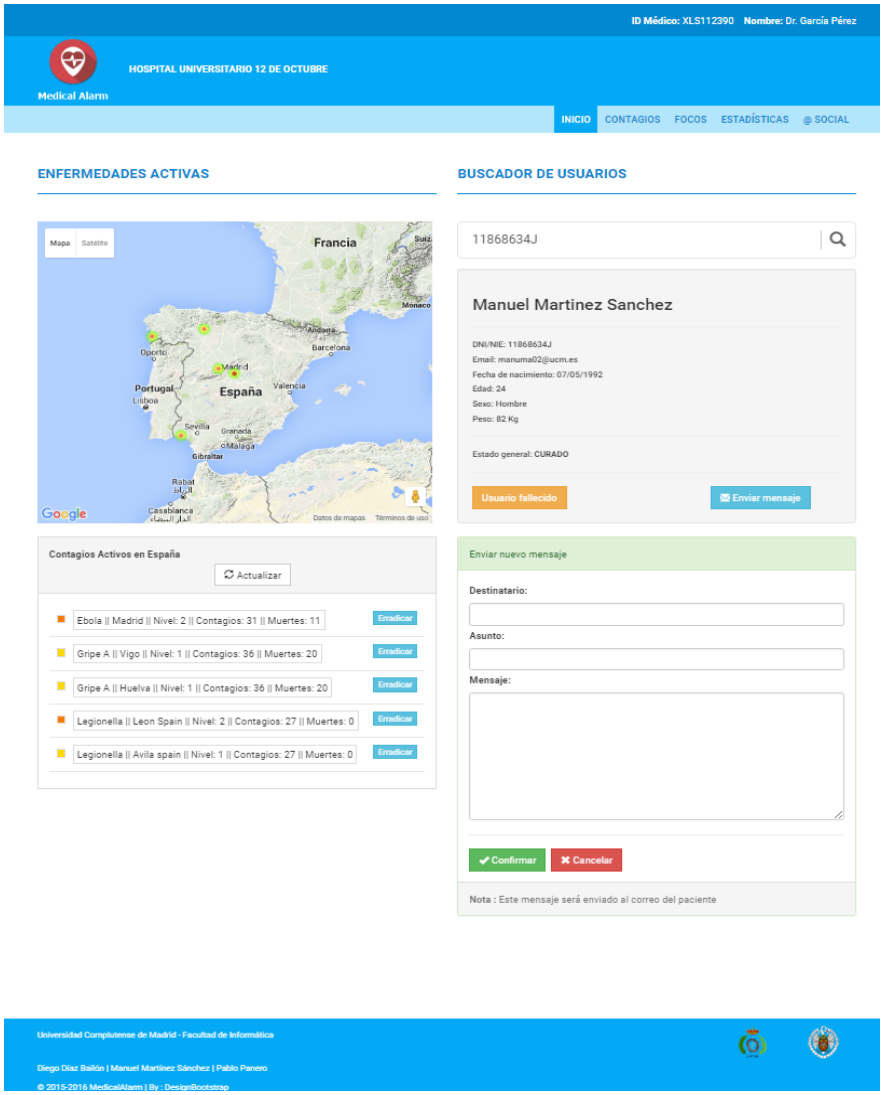
La última pestaña, correspondiente a Datos, es un estudio exhaustivo a través del análisis y técnicas de Big Data del sistema, extrayendo e infiriendo cifras y porcentajes sobre los pacientes y los lugares, siendo capaz el sistema de mostrar, mediante gráficas, diagramas de dispersión, aproximaciones, medias y datos para la prevención de enfermedades y el control de las mismas.



62 - Figura 6.2.4 - Prototipo Datos

Con todos los prototipos completos, el siguiente paso en el proceso fue ofrecer al usuario la posibilidad de analizar el prototipo y ver su reacción. Se consiguió contactar con un médico del Hospital Universitario 12 de Octubre, del departamento de infecciosas, que se ofreció para evaluar la idea. Se obtuvo mucha información útil para desarrollar la web y otorgarle el toque formal-científico del que carecía. Haciendo uso de librerías gráficas y

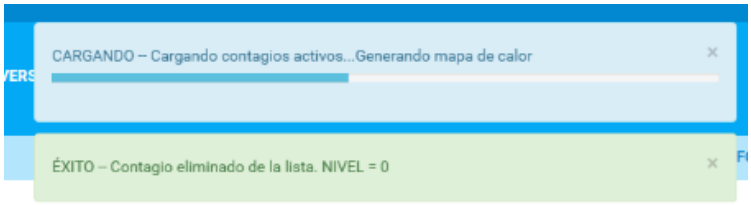
siguiendo los consejos de los médicos se llevó a cabo la página web que, gracias al framework *Bootstrap*, quedo de la siguiente manera:



63 - Figura 6.2.5 - Diseño web final

Se muestra la portada y vista principal de la web, donde se aprecian los patrones y principios explicados anteriormente, como el principio de proximidad o consistencia mediante el sistema de pestañas, resaltando la actual, o mediante la disposición de los botones y buscadores, situados en las esquinas donde es más rápido acceder a ellos. Todos los elementos

son Bootstrap y se ha seguido el principio de feedback y estado visible de la aplicación incorporando, entre otros elementos, barras de estado en el proceso de carga de los mapas, las gráficas o la actualización de los datos.

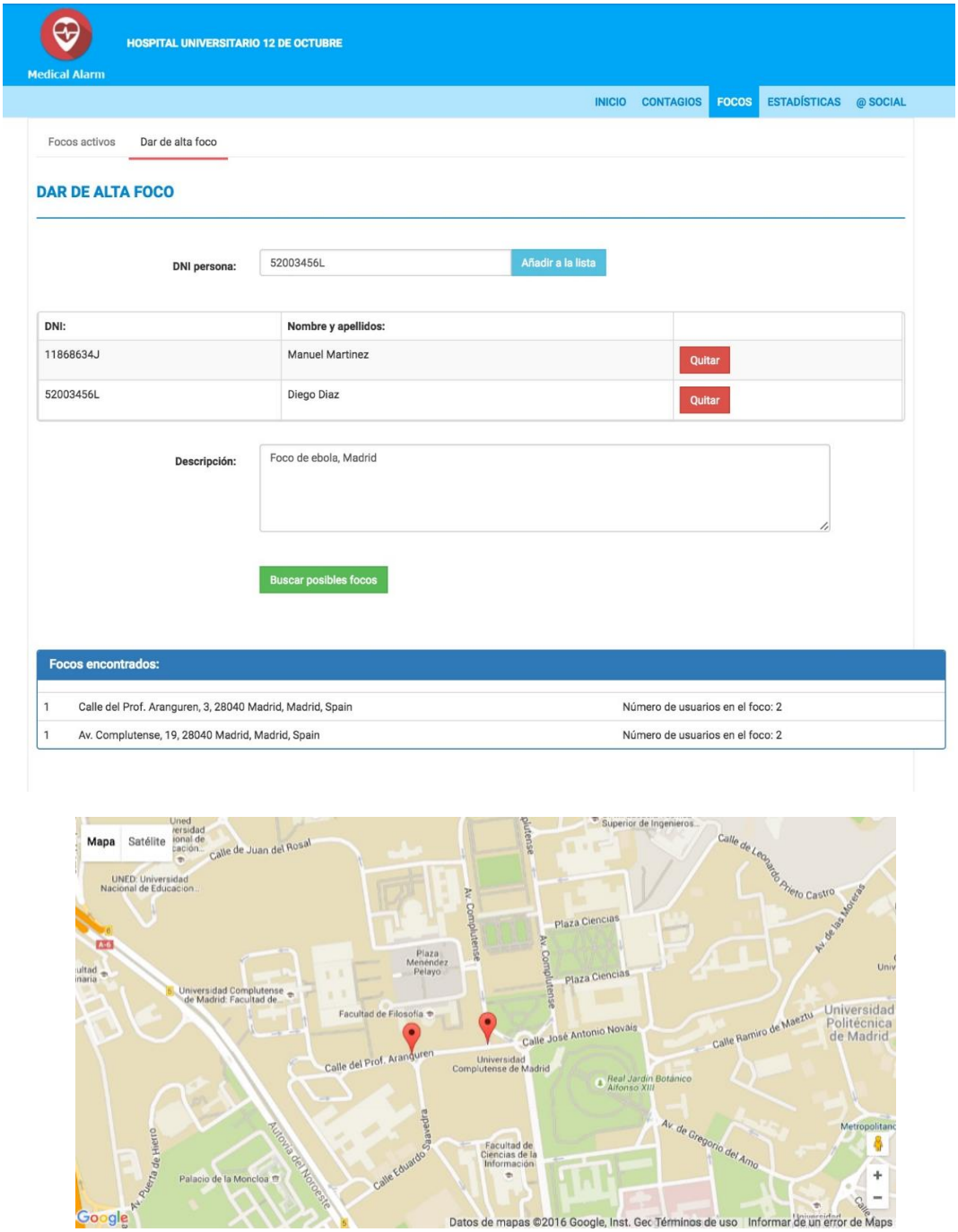


64 - Figura 6.2.6 - Feedback web

La página web contiene una pestaña con gráficas y tablas que contienen los datos recogidos de las bases de datos con las que se relaciona. Estas visualizaciones son de vital importancia para un médico, por lo que se han diseñado prestando especial interés a la disposición, tamaño y acciones posibles. El resultado, gracias a una serie de librerías, ha sido el siguiente:

65 - Figura 6.2.7 - Gráficas

Por otro lado, el principio de libertad del usuario se puede apreciar en el sistema secundario de pestañas, dándole libertad para navegar por la web y con botones para poder deshacer las acciones. Toda la información se muestra en formato de tablas y recuadros para ayudar a la memorabilidad y a la facilidad de aprendizaje.



66 - Figura 6.2.8 – Focos

Capítulo 7. Implementación

Para una buena comprensión de cada módulo del sistema, se detalla a continuación la implementación de cada una de ellos.

7.1 Implementación Android

Como en todo proyecto Android, se cuenta con tres paquetes principales: manifest, java y res. En el primero de ellos, en el paquete '*manifest*', se encuentra el *AndroidManifest.xml*, un archivo de configuración situado en la raíz de la aplicación donde se aplican las configuraciones básicas de la misma. En él se declaran tanto las actividades y los servicios implementados, como los permisos que necesitará la aplicación para su funcionamiento, que en este caso son:

- Permisos para acceder a internet:

<uses-permission android:name="android.permission.INTERNET" />

- Permisos para acceder y manejar el Account Service:

<uses-permission android:name="android.permission.GET_ACCOUNTS" />

<uses-permission android:name="android.permission.MANAGE_ACCOUNTS" />

<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />

<uses-permission android:name="android.permission.USE_CREDENTIALS" />

- Permisos para obtener la localización del dispositivo:

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

Por su parte, el paquete '*res*' es el que contiene los recursos de la aplicación, tanto las imágenes (carpetas 'drawable' o 'mipmap'), como los ficheros XML de las vistas (carpeta 'layout') o los ficheros XML donde se recogen ciertos valores usados en la aplicación permitiendo cambiarlos sin tener que indagar en el código fuente (carpeta 'values'), etc.

Y por último, en el paquete *'java'* es donde se encuentra el grueso del código, es decir, todas las clases y paquetes que se han implementado para crear la aplicación. En este caso se cuenta con las clases de las vistas y los fragments/activities y, además, con cuatro paquetes para encapsular y definir mejor las funcionalidades de la aplicación. El paquete 'apiConnections' es el encargado de realizar las tareas relacionadas con la conexión con las bases de datos; el paquete 'models', usado para indicar la estructura de los modelos que se necesitan; el paquete 'sensors', que contiene las clases necesarias para la obtención y tratamiento de datos de los sensores; y, por último, el paquete 'commons', en el que se encapsulan funcionalidades útiles y comunes para toda la aplicación, como por ejemplo la implementación de un *Lock* que fue necesaria para el correcto tratamiento de datos de los sensores y que, quizás, pueda ser de utilidad en un futuro.

Una vez conocida la estructura interna de la aplicación, se pasa a explicar más en detalle la implementación de cada una de las funcionalidades descritas. Para poder entenderlas correctamente y no tener que repetirlo en cada epígrafe, es importante explicar que todas las peticiones al API que se mencionen se realizan de manera asíncrona, es decir, se crea otro hilo de ejecución paralelo al principal. El hilo o 'thread' principal en Android es el que se encarga de las vistas; por tanto tareas como peticiones al API o a las bases de datos, que a priori no se sabe cuánto van a tardar, se deben realizar de manera paralela al thread principal para evitar bloquearlo y que el usuario se frustre, ya que la aplicación visualmente queda bloqueada.

Todas estas tareas asíncronas se han implementado creando clases internas que extienden la clase abstracta *AsyncTask* y, a su vez, las peticiones al API se han llevado a cabo mediante el objeto *HttpURLConnection*.

Ahora sí, se procede a la explicación individual de cada funcionalidad:

IMPLEMENTACION 1: Envío de la localización

El sistema utiliza los valores de varios sensores: acelerómetro, magnetómetro, luminosidad, localización y estado de la batería. Todos ellos son obtenidos a través de un escuchador implementado para cada sensor que actualiza los valores correspondientes cuando detecta un cambio.

En el caso de los sensores correspondientes a magnetismo, aceleración y luminosidad los valores son obtenidos implementando la interfaz "*SensorEventListener*" de Android que produce un evento y notifica al escuchador en el caso de un cambio.

En el caso del sensor correspondiente al estado de la batería se implementa la interfaz "*BroadcastReceiver*", que actualiza periódicamente el estado de la misma; aunque el estado de

carga no cambie, cambia la cantidad de carga de la batería y por ello obtenemos una actualización. Sin embargo, el único valor que se guardará será el estado: "cargado desde corriente alterna", "cargando desde usb" o "descargando".

En cuanto a la localización geográfica, el servicio de sensores hace uso de las librerías de Google, por lo que es necesario que el dispositivo móvil tenga activados los servicios de Google Play y el GPS. En este caso se implementa la interfaz "*LocationListener*". Además se especifica la opción "*PRIORITY_BALANCED_POWER_ACCURACY*" para evitar un consumo excesivo de batería de la aplicación en el dispositivo. La información sobre el posicionamiento geográfico se actualizará con una periodicidad de 5 minutos, salvo que se produzca una actualización antes de que pase este tiempo (por ejemplo en el caso de que otra aplicación realice una petición de actualización de la localización).

Todos los sensores anteriormente explicados son utilizados por un servicio que corre en segundo plano en el dispositivo para no realizar un uso inadecuado del hilo de la interfaz gráfica. Dicho servicio actualizará los datos cuando reciba actualizaciones de los distintos sensores y encapsulara los datos en un objeto de tipo "*SensorResult*".

En lo que respecta al envío de todos los valores recogidos, se ha usado de nuevo un servicio, pero en este caso un *IntentService*. La peculiaridad de este tipo de servicio es que se ejecuta en segundo plano, lo cual es útil para evitar bloquear la interfaz. Así pues, cuando se han detectado cambios en los sensores explicados arriba, se lanza este servicio pasándole los nuevos valores y es esta clase la que se encarga de realizar la petición a la API, mandando dichos valores en formato JSON.

/malarm/api/sensors/

IMPLEMENTACION 2: Consulta del estado actual del usuario y de la zona

Para poder mostrar al usuario su estado actual y el de su zona, lo primero que se debe hacer es obtener el DNI que identifica al usuario gracias al objeto *SharedPreferences*; este es un objeto que proporciona Android para poder almacenar y recordar valores como el idioma de la aplicación o el usuario/password, para, de esta manera, no obligar al usuario a logearse cada vez que abra la aplicación.

Una vez obtenido el DNI, se necesitan las coordenadas del dispositivo, para ello se lanza el servicio que se ha implementado de los sensores y se toman sus valores actuales. Cuando el servicio proporciona dichos valores, ya se puede realizar la petición al API de manera asíncrona como se explicó en la introducción de este capítulo:

/malarm/api/user/status/?user_dni=12345678J&lat=40.451816&lng=-3.729581

La respuesta a esta petición es un objeto JSON que contiene la lista de los contagios que tiene el usuario y la lista de los contagios que existen en la zona (por supuesto estas listas pueden estar vacías en caso de no existir contagios). Así pues, una vez parseada correctamente la respuesta, solo queda actualizar los campos e imágenes de la vista para ofrecerle el feedback al usuario.

Además de lo recientemente explicado, también se ha implementado la funcionalidad 'swipe' de la vista. Con 'swipe' se hace referencia al gesto de deslizar la pantalla hacia abajo, es una acción ya estandarizada en Android y que permite actualizar tanto los valores del estado actual del usuario como de la zona en la que se encuentra. Para ello simplemente se engloba la vista dentro de un objeto *SwipeRefreshLayout* e implementando su método *setOnRefreshListener()*, se consigue que se vuelva a ejecutar todo lo explicado anteriormente.

IMPLEMENTACION 3: Consultar y eliminar noticias

Al igual que en la funcionalidad anterior, lo primero que se necesita es el DNI del usuario, el cual se obtiene de SharedPreferences del mismo modo, y se realiza una vez más, de forma asíncrona, la petición al API:

/malarm/api/news/?user_dni=12345678J

La respuesta vuelve a ser un objeto JSON que se debe parsear para obtener las noticias y sus descripciones, con las cuales crear la lista. Ésta, por su parte, está formada por tarjetas o 'cards' que contienen la información de la noticia. Para ello se crea la clase *CardInfo*, que contiene todos los campos que la forman, y el modelo *New*, que contiene los campos de la noticia; de esta manera se puede trabajar con ellas de forma mucho más sencilla y aislada.

Además de visualizarlas, también se permite al usuario eliminar una noticia en caso de que así lo quiera. Para ello se añade en la parte derecha de cada 'card' una cruz mediante la cual la noticia desaparecerá. La forma más correcta de implementarlo es englobar la vista de la lista en un objeto *RecyclerView* y extender *RecyclerView.Adapter* en la clase *CardAdapter*, la cual se encarga de mantener actualizada la lista de la vista. De esta manera se facilita bastante la tarea, ya que simplemente se almacena la lista de noticias en CardAdapter y en caso de presionar la cruz se ejecuta el listener que elimina la noticia correspondiente y automáticamente actualiza la vista.

Para terminar con esta funcionalidad, solo queda mencionar que, al igual que en el caso anterior, se ha implementado el *SwipeRefreshLayout* para actualizar con un simple gesto todos los valores de la vista.

IMPLEMENTACION 4: Consultar perfil

Como en los dos casos anteriores, se necesita del DNI para poder obtener los datos del usuario y, de nuevo, se requiere del ya mencionado objeto SharedPreferences para obtenerlo. Siguiendo con la metodología, se realiza la petición al API que, nuevamente, devolverá la respuesta en formato JSON con todos los datos que se deben mostrar al usuario (nombre, apellidos, fecha de nacimiento, peso, estatura, etc.).

/malarm/api/user/12345678J

Para ayudar en la construcción de los datos del usuario fue útil la creación del modelo *User* en el paquete 'models' de la aplicación, el cual únicamente contiene los campos del usuario y un método estático para su construcción a partir del objeto JSON.

IMPLEMENTACION 5: Cambiar el idioma

En esta funcionalidad es clave el paquete 'res' mencionado al principio del capítulo, ya que en él se encuentra una carpeta que contiene los ficheros XML que recogen los valores *string* que se usan en la aplicación. Existe un archivo *string.xml* para cada idioma al que se haya querido traducir la aplicación (en este caso del español y del inglés) y es en ellos donde se crean las cadenas de texto, identificadas por un nombre único. De esta manera no se debe modificar nada del código fuente para la traducción, únicamente se debe saber qué archivo de idioma cargar y automáticamente la aplicación sustituirá cada referencia a una cadena de texto por su texto correspondiente.

La primera vez que se ejecute la aplicación se tomará como idioma seleccionado el que esté por defecto en el dispositivo, pero en caso de que el usuario desee cambiarlo, éste se guardará en el objeto SharedPreference de la aplicación.

Explicándolo más en detalle, lo primero que se hace al iniciar la aplicación es comprobar si en SharedPreferences existe un valor guardado para el idioma; si no existe no es necesario cambiar nada, pues Android por defecto automáticamente toma el string.xml del idioma del dispositivo. En caso de que si exista un valor almacenado, se debe modificar el objeto *Locale* de la aplicación; ésta es una clase proporcionada por Android que se encarga de trabajar con temas de configuración de la aplicación, en este caso el lenguaje. Para ello simplemente se crea e inicializa un objeto Locale auxiliar con el idioma que esté almacenado en las preferencias, y se establece como 'default' para la configuración de la aplicación.

Así pues, solo queda explicar cómo se ha implementado la forma mediante la que el usuario podrá realizar esta tarea. La forma escogida fue mediante un ListPreference, algo similar a una lista de checkbox de Java, de forma que solo se pueda seleccionar uno de ellos. La

manera más correcta de implementar esto en Android es estableciendo los valores de las opciones en los archivos *array.xml* de la carpeta values (dentro del paquete 'res'), ya que también existe un archivo para cada idioma e implementando el listener del *ListPreference*. Simplemente es necesario que, cuando se detecte la pulsación de una de las opciones, se establezca ese valor del lenguaje en las preferencias y se reinicie la aplicación con el nuevo idioma.

IMPLEMENTACION 6: Login

El login es la primera funcionalidad que ve el usuario; sin embargo, solo se le muestra la primera vez que abre la aplicación o en caso de que haya borrado los datos de preferencias. Una vez más se necesita el objeto SharedPreferences, en este caso para guardar el usuario y la contraseña del usuario, de esta manera se evita mostrarle el login al usuario cada vez que abra nuestra aplicación. Simplemente se debe comprobar si dicho objeto contiene las preferencias guardadas, y solo en caso de no ser así lanzar la actividad del Login.

Como ya se ha mencionado, para logearse se le solicita al usuario que introduzca su usuario y su contraseña. Una vez introducidos y pulsado el botón de 'Entrar', se realiza la petición al API para que devuelva el usuario identificado con ese mismo DNI. Ahora pueden pasar tres cosas: que no se encuentre a ningún usuario con el DNI introducido, con lo cual se muestra un error; que el usuario sí exista pero las contraseñas no coincidan, también se muestra un error; o que tanto el usuario como la contraseña sean correctos, en cuyo caso se guardan en SharedPreferences para posteriores usos y se deja que continúe la normal ejecución de la aplicación.

Por último, para aquellas personas que no dispongan aún de una cuenta, disponen de un botón en la parte inferior de la vista del login que le permitirá crearse una, se explica justo a continuación.

IMPLEMENTACION 7: Registro

Al igual que en el caso del login, el registro también está implementado como una actividad, la cual es lanzada cuando se presiona el botón de 'Crear cuenta' de la vista del login.

En este caso se solicitan al usuario todos los datos que se necesitan para rellenar su perfil y, tras las validaciones pertinentes de los mismos (campos no vacíos, comprobar que el email tiene el formato correcto, que la password y el re-password coincidan, etc.), se crea el parámetro JSON que se enviará junto a la petición al API. A diferencia de todos los casos

anteriores, ahora se debe especificar que se requiere del método POST de HttpURLConnection para realizar la petición.

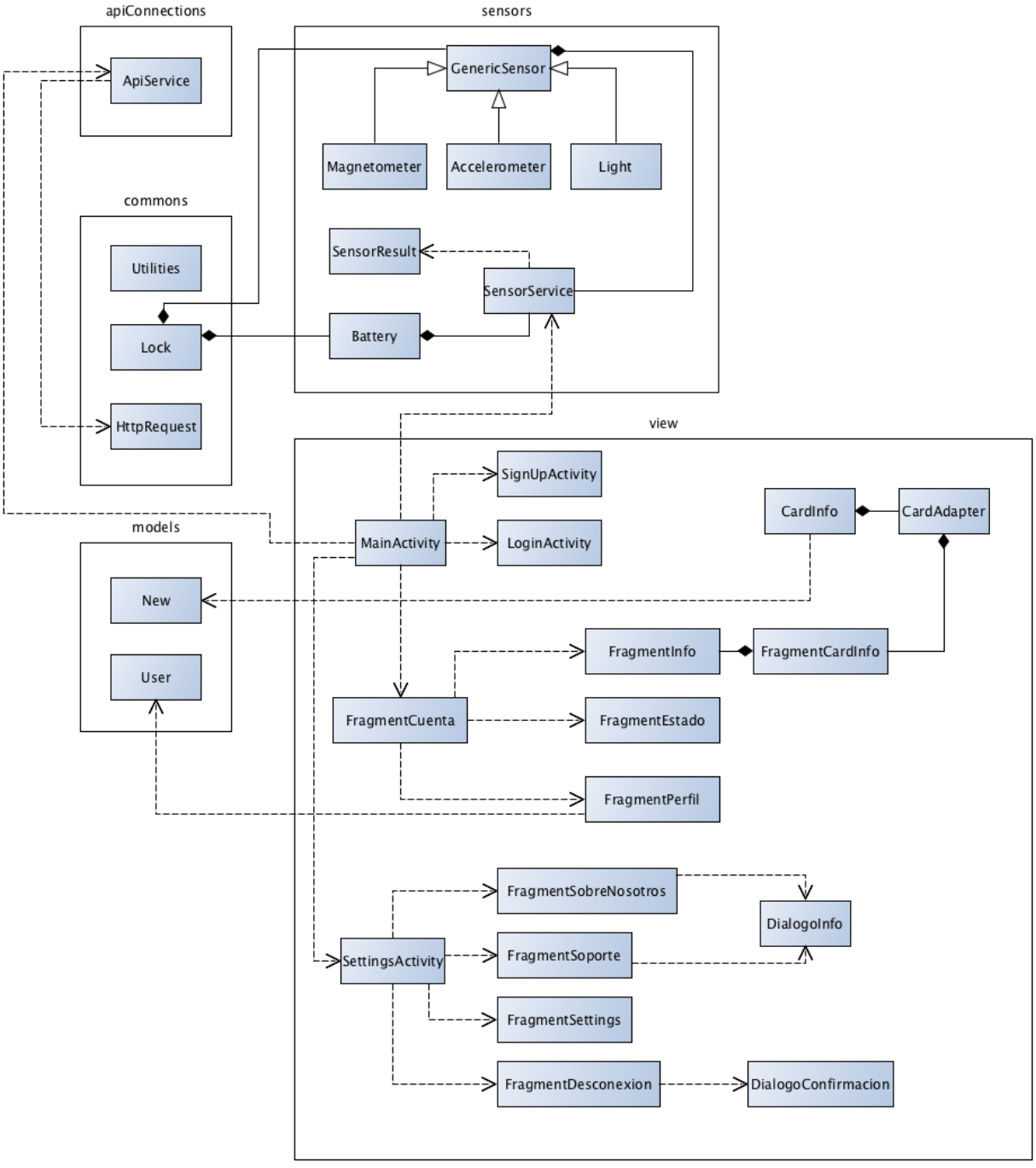
IMPLEMENTACION 8: Activar/desactivar envío de datos de los sensores

Al crear el objeto *FragmentSettings* se enlaza ("*bind*") con el servicio encargado de obtener y actualizar los datos de dichos sensores. Al activar o desactivar los datos de los sensores se comprueba si los sensores están activos o no, obteniendo las variables "*sensor_status*" y "*sensors_initialized*" del SharedPreferences y cuál es el nuevo estado siendo las posibles transiciones las siguientes:

Estado de los sensores	Activar/Desactivar	Nuevo estado sensores
Activos	Activar	Activos
Activos	Desactivar	Desactivados
Desactivados	Activar	Activos
Desactivados	Desactivar	Desactivados

67 - Tabla 7.1.1 - Transiciones sensores

En el caso de que los sensores hayan cambiado de estado, se actualizan las correspondientes variables "*sensor_status*" y "*sensors_initialized*" en el objeto SharedPreferences.



68 - Figura 7.1.1 - Diagrama clases Android

7.2 Implementación web

Actualmente, parece que la implementación de una página web no debería ser muy tediosa, ya que existen gran cantidad de herramientas que agilizan el trabajo y además los lenguajes web parecen muy extendidos por todo el mundo de Internet.

En la web, mediante peticiones *Ajax*, la comunicación con la API se realiza a través de la url que corresponda. Ésta devuelve la información solicitada a través de un JSON por lo que en la página web sólo es necesario procesar ese JSON mediante el uso de Javascript.

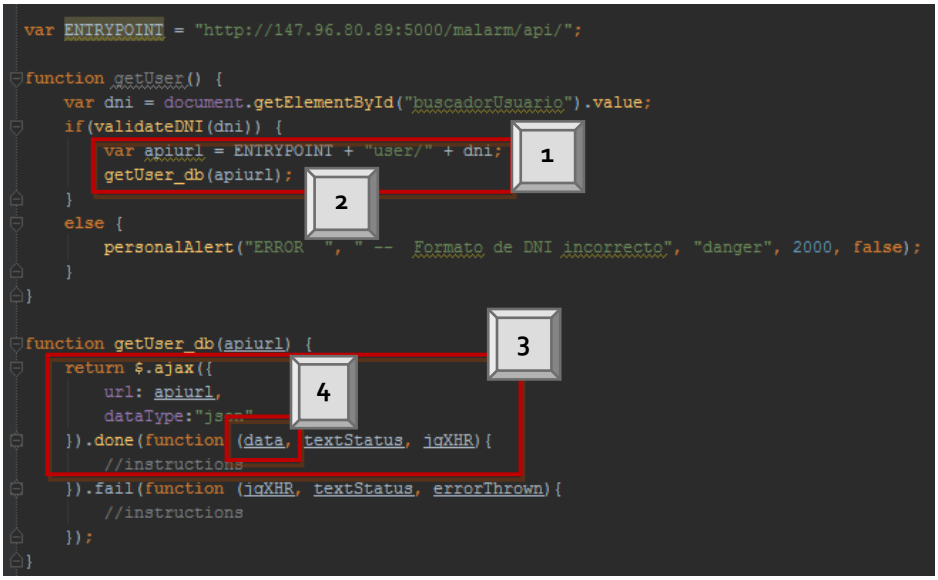
Resumiendo, la existencia de una API hace que la página web varíe así como el tratamiento de los datos y la conexión con las bases de datos. La web se ha desarrollado siguiendo el framework *Bootstrap* haciendo uso de *HTML5*, *CSS3*, *JavaScript*, *jquery* y *Ajax*. Para que la comprensión de la conexión y la comunicación con la API quede más clara, antes de explicar en profundidad la implementación de cada funcionalidad, se detalla este proceso.

Lo primero que hay que tener claro es la url a la que conectarse. Un ejemplo podría ser conectarse a local (localhost) y pedir un usuario en particular con un DNI:

```
var ENTRYPOINT = "http://localhost:5000/malarm/api/user/11868634J";
```

69 - Figura 7.2.1 - URL

El objetivo del javascript va a ser crear estas urls para solicitar al api la información deseada y poder así mostrar toda la información devuelta en la web. Se puede resumir todo el proceso en parsear y crear la url correcta (1) y parametrizada mandársela a otra función (2) encargada de hacer la petición Ajax (3) la cual devuelve la información en el parámetro data (4) en formato JSON. Siguiendo con el ejemplo anterior, se muestra el proceso completo para solicitar la información de un usuario:



70 - Figura 7.2.2 - Petición al API

Una vez explicadas las tecnologías usadas, y aclarado uno de los puntos singulares de la arquitectura como es el API, se aclara que la web tiene una estructura que mediante carpetas, separa los ficheros html del resto (*assets*) diferenciándose los ficheros css (css), los javascript (js) y las imágenes (img). Por último, para la comprensión total de la implementación de la página web se pasa a detallar cómo se ha llevado a cabo cada funcionalidad ofrecida.

La web cuenta con dos elementos estáticos y un sistema de pestañas para la navegación, diferenciándose Inicio, Contagios, Focos, Estadísticas y @Social. El primer elemento estático es el header, común para todas las pestañas y el cual contiene la información relacionada con el médico y el hospital en el que se encuentra y el logo de la web. Actualmente, no existe en la web una identificación y una administración de médicos por lo que se refleja un nombre estático como plantilla al igual que el hospital. Las bases de datos están diseñadas e implementadas para soportar un registro de médicos pero de momento, no se ha implementado en la web quedando reservado y anotado en el trabajo futuro. El segundo elemento es el footer, común también para todas las pestañas y donde se puede observar toda la información relacionada con los desarrolladores del proyecto, datos de contacto y universidad a la que pertenecen, la Universidad Complutense de Madrid. A continuación, se hace un recorrido por cada una de las pestañas deteniéndose en sus funcionalidades.

Se comienza por la pestaña de Inicio, la pestaña que se muestra al cargar la página y por lo tanto portada de la web. En ella se encuentra:

IMPLEMENTACIÓN 1: Ver contagios activos en España

Para que el médico visualizara de una manera cómoda y rápida todos los contagios activos por el momento, se optó por un mapa de calor. Gracias al API de Google Maps y haciendo uso de javascript, se fijo en la web un mapa de España (map = new google.maps.Map("map")) seteándole como puntos de calor los contagios activos recibidos tras la petición Ajax correspondiente. Internamente los algoritmos y las búsquedas utilizan coordenadas exactas con una latitud y una longitud pero a la hora de mostrar en el mapa y en su correspondiente leyenda había que mostrar la zona en la que se encontraban esas coordenadas. No servía mostrar [40.401703, -3.720495], coordenadas de un lugar de Madrid, sino que se debía mostrar Madrid. Para solucionar este problema se optó por implementar las clases *GeoCoding* y *ReverseGeoCoding*, las cuales dadas unas coordenadas devolvían la zona o viceversa. Gracias a estas clases se pudo mostrar el mapa de calor correctamente con su leyenda en la parte inferior recogiendo los datos principales de cada contagio activo dibujado en el mapa. La función que permite realizar todo esto pide a la API (GET) los datos mediante esta url que devuelve en formato JSON el conjunto de contagios:

/malarm/api/contagions?type=contagions

IMPLEMENTACIÓN 2: Actualizar contagios activos de España

Si no se ha actualizado la página y se ha producido algún cambio relacionado con los contagios activos, existe un botón Actualizar que permite sin necesidad de tener que recargar toda la página refrescar los contagios activos. De esta manera se evitan cómputos y cargas pesadas como la carga de todo el mapa. La implementación y el código que hay por debajo es similar a la funcionalidad anterior pues la llamada a la API (GET) es la misma; simplemente se trata de un botón que repite una acción determinada, la de actualizar los datos de los contagios.

IMPLEMENTACIÓN 3: Erradicar un contagio activo

Dentro de la leyenda donde se encuentran todos los contagios activos, si se pulsa el botón Erradicar se lleva a cabo una petición a la API (DELETE) eliminando el contagio activo y actualizando la lista actual. Haciendo uso de jquery y javascript se puede controlar y manejar el árbol DOM y mediante *parentNode* y *removeChild*, eliminar el elemento de la lista seleccionado.

/malarm/api/contagion/id/

IMPLEMENTACIÓN 4: Buscar un usuario en el sistema

Es normal encontrarse con buscadores en las páginas web. En este caso existen dos, uno de ellos el buscador de pacientes por DNI situado en esta pestaña en la parte superior derecha. Se ha cuidado la comprobación de la entrada, permitiendo sólo la búsqueda de DNIs reales con el formato correcto. Asegurada una entrada correcta, se solicita a la API (GET) el usuario mediante el DNI introducido y se muestran todos los datos del paciente. Mientras no se haya buscado ningún usuario, el espacio de información permanece vacío y gracias al atributo *hidden* y *show* en css, se puede llevar a cabo un vistoso efecto en el que se cambia el contenido del recuadro.

*/malarm/api/*user*/dni/*

IMPLEMENTACIÓN 5: Cambiar estado de usuario a fallecido

Dentro del buscador de un usuario, una vez mostrada la información del usuario solicitado, gracias a las propiedades anteriormente mencionadas de *show* y *hidden*, aparecen dos botones, Usuario fallecido y Enviar mensaje. Al pulsar en el botón de Usuario fallecido, simplemente se lleva a cabo una actualización del estado del usuario, un simple update. Por lo tanto, se trata de una implementación en la que se lleva a cabo una petición PUT a la API con el usuario mostrado con la misma url que el caso anterior.

IMPLEMENTACIÓN 6: Eliminar enfermedad de un usuario

Una vez que se ha buscado un usuario en el sistema y que la búsqueda ha tenido éxito, se muestra el paciente. En caso de que su estado sea enfermo, y sólo en ese caso, se muestra la lista de enfermedades de las que el usuario está contagiado. En esta lista, para cada enfermedad, existe un botón verde de Curar que permite al médico eliminar esa enfermedad

del paciente en caso de que se haya curado. Al eliminar una enfermedad del paciente, en el sistema hay que reflejarlo en varios lugares. Primero hay que actualizar la vista suprimiendo la enfermedad, pero al curarse se debe, mediante la API (PUT), actualizar las estadísticas y los datos referentes al total de contagiados, curados y otros datos estadísticos. Por último, hay que controlar si, después de curar una enfermedad, quedan más enfermedades en ese paciente ya que en ese caso, el estado se mantiene enfermo; pero si ya no tiene enfermedades, el nuevo estado será curado.

/malarm/api/users/contagions

IMPLEMENTACIÓN 7: Mostrar notificaciones enviadas a un paciente

Al igual que en funcionalidades anteriores, lo primero que se debe hacer es verificar que el DNI introducido es válido; una vez comprobado, se realiza la petición al API solicitándole las notificaciones de dicho usuario. Tanto si la petición falla como si es satisfactoria, se debe vaciar la tabla que estuviera previamente. En caso de que la petición no falle, es decir, que devuelva la lista de notificaciones del usuario, se modifica el código HTML, creando tantas filas como notificaciones se hayan recibido.

Además, como se explicará en la siguiente funcionalidad, se podrá confirmar si el usuario efectivamente esta contagiado o si, por el contrario, se trata de una falsa alarma. Ambas opciones se encuentran en forma de dos botones en la tabla que se acaba de crear, así que para saber qué usuario y de qué contagio se trata, se establece como id's de los botones el id del usuario y el id del contagio correspondiente, para que cuando se pulse uno de ellos se pueda realizar la pertinente función sin necesidad de realizar más peticiones al API.

IMPLEMENTACIÓN 8: Confirmar contagio de usuario

Como se acaba de mencionar en el apartado anterior, en la tabla se tienen dos botones, uno para confirmar si el usuario esta contagiado por la enfermedad de la cual se le envía la notificación, y otro para confirmar si ha sido una falsa alarma. Así pues, esta funcionalidad va ligada al primero de los botones que, al pulsarlo, ejecuta una función que toma los valores almacenados en el id del botón (el id del usuario y el id del contagio) y la fila que va a desaparecer a raíz de eliminar esa notificación del usuario. Por tanto, se realiza la correspondiente petición al API, pasándole dichos valores en formato JSON, y estableciendo que se trata de una petición de tipo 'PUT', ya que realmente lo que se hace es modificar el valor 'confirmado' de la notificación.

Lo siguiente a hacer es, por supuesto, establecer que el usuario está efectivamente contagiado, por lo que se debe realizar una nueva solicitud al API, en este caso de tipo 'POST', ya que únicamente lo que se debe hacer es añadir al usuario a la lista de usuarios contagiados por el contagio pertinente.

Y, por último, solo queda actualizar la tabla para que elimine la fila de la notificación que se acaba de confirmar. Para ello se modifica el árbol DOM del documento HTML borrando la fila que es recibida al principio de la explicación.

IMPLEMENTACIÓN 9: Confirmar falsa alarma de un posible usuario contagiado

Esta funcionalidad corresponde al segundo de los botones de la tabla de notificaciones y comparte cierta implementación con la anterior, por ejemplo, en este caso también obtenemos el id del usuario y del contagio y la fila de la tabla y realizamos la petición 'PUT' al API con dichos valores.

A diferencia de la funcionalidad anterior, ahora no se debe añadir el usuario a ninguna lista, simplemente se indica que ha sido una falsa alarma y que, por tanto, el usuario no se ha contagiado, así que lo único que queda por hacer es eliminar, igual que antes, la fila correspondiente.

IMPLEMENTACIÓN 10: Dar de alta un contagio

Siempre que se da de alta un contagio o un foco, es importante realizar una comprobación de los valores que ha introducido el usuario para evitar errores o entradas malintencionadas. En este sentido se comprueba que no haya campos vacíos y solo se permite la entrada de aquellos valores válidos para el campo que se esta solicitando, por ejemplo, para el tiempo de exposición solo se permite la entrada de números y para introducir la enfermedad se muestra la lista de enfermedades disponibles, para que el usuario solo tenga que seleccionar cual quiere.

Así pues, una vez realizadas estas comprobaciones, se crea el objeto JSON a partir de los valores tomados del formulario y se envía junto a la petición 'POST' al API.

Es importante aclarar que la lista de enfermedades que se le proporciona al usuario para que dé de alta el contagio se corresponde con las enfermedades existentes en el sistema. Por tanto, para rellenar dicha lista en el formulario, se realiza una petición más al API solicitándole todas las enfermedades almacenadas en la base de datos.

IMPLEMENTACIÓN 11: Ver focos activos

Lo primero que se debe hacer es realizar la petición al API para que se devuelva la lista de todos los focos activos existentes; una vez obtenida la respuesta, se rellena la tabla de los focos. Al igual que en las funcionalidades anteriores, se usa el método de guardar en el id del botón 'Eliminar foco' el lugar y el id del foco, de esta manera cuando se quiere eliminar un foco, se cogen estos valores y se realiza la petición necesaria.

IMPLEMENTACIÓN 12: Eliminar foco

La presente funcionalidad se encuentra ligada a la anterior, ya que, en la tabla que se genera con los focos existentes en el sistema, se añade un botón para que el médico o especialista pueda eliminarlo cuando se haya extinguido.

La manera en la que se ha implementado ha sido, como se ha mencionado antes, almacenando en el atributo 'id' del botón el lugar del foco y el id del foco (ya que un mismo foco puede estar en dos lugares diferentes), así que de esta manera cuando es presionado el botón se toman estos dos valores y se realiza la petición 'DELETE' correspondiente para dar de baja ese foco.

Por último, al igual que en funcionalidades anteriores, solo queda actualizar la tabla eliminando la fila del foco que se ha extinguido.

IMPLEMENTACIÓN 13: Dar de alta foco

Se trata de una de las funcionalidades más importantes de la web junto con dar de alta un contagio; para dar de alta un foco se necesitan al menos dos usuarios para poder saber donde se pudieron cruzar y donde existirían posibles focos. En este sentido, lo primero que se debe hacer es, una vez más, comprobar si el DNI introducido es válido, realizar la petición al API para obtener el usuario, y añadirlo a la tabla de la cual el algoritmo tomará los usuarios. Además se ha añadido un botón, para poder quitarlo de la lista en caso de que el médico así lo desee o se haya podido confundir.

Así pues, una vez se pulse el botón de dar de alta, se comprobará que al menos existan dos usuarios en la tabla y que el campo de la descripción no esté vacío, ya que es un campo requerido y, si todo esto es correcto, realizar la petición 'POST' con el objeto JSON que contiene todos los valores necesarios para el proceso de alta (los DNIs de los usuarios y la descripción).

Una vez que se haya ejecutado la petición y, por tanto, el algoritmo de búsqueda de focos, devolverá la lista de lugares donde existan posibles focos y el número de usuarios que coincidieron en ese punto; es decir, si por ejemplo se dio de alta un foco con cinco usuarios para ver donde se cruzaron, a lo mejor en un punto se cruzaron cuatro de ellos y en otro solo dos, y éste es un dato que puede ser de gran interés para el médico. Explicado esto y volviendo a la funcionalidad, se crea una tabla con los lugares y el número de usuarios en el foco y, justo debajo, se añade un mapa donde, gracias al API de *Google Maps* se pintan los puntos en las localizaciones obtenidas. Para ello se crea un mapa centrado en Madrid (para que aparezca toda la Península Ibérica) y se va recorriendo el listado de puntos creando *'markers'* (iconos) en esa latitud-longitud. Por último, se ha implementado el evento *'listener'* del marker para que, cuando el médico pulse sobre el icono, le aparezca el número de personas que se encuentran en ese foco.

IMPLEMENTACIÓN 14: Eliminar usuario de la lista de foco

Como se mencionó en la funcionalidad anterior, se pensó que, si el médico se equivoca al introducir el DNI de un usuario o quiere modificar el alta, se le debe proporcionar un método para ello. Así pues, se añadió a la derecha de cada usuario de la tabla un botón para quitarlo de la misma, de modo que, si es presionado, se recoge el número de la fila y se elimina de la tabla.

En la pestaña de Estadísticas se encuentran dos subpestañas, Datos generales y Enfermedades con detalle.

IMPLEMENTACIÓN 15: Visualizar datos generales

En la pestaña de Datos generales no hay mucha interacción con el usuario. Se trata de una pestaña de visualización en la que se puede consultar en forma de gráfica datos sacados del sistema como los dos rankings de enfermedades más mortales o enfermedades más infecciosas y una gráfica que relaciona el número de personas y cada estado posible (indefinido, sano, enfermo, curado, fallecido). Lo más importante de esta pestaña es la creación de las gráficas y la forma en que se muestran los datos. Para ello se ha utilizado la librería *c3.js* especial para gráficas, la cual facilita enormemente el tratamiento de los datos. Permite crear las gráficas a partir de un fichero JSON que se obtiene a través de la API (GET). Todo se controla en javascript al crear las gráficas con una serie de campos: *data, axis, size, bindto,* etc. Gracias a *c3.js* se ha podido dar un formato formal y profesional a las gráficas e

internamente sólo hay que realizar una serie de llamadas a la API ya que todo el cómputo y las consultas son ajenas a la web, flask y Python se encargan de ello.

*/malarm/api/*diseases/?type=c&top=5 (ranking más infecciosas)

*/malarm/api/*diseases/?type=d&top=5 (ranking más muertes)

*/malarm/api/*diseases/users/?type=status (número de usuarios en cada estado posible)

IMPLEMENTACIÓN 16: Generar informe general

En la pestaña de Datos generales existe un único botón el cual permite exportar todas las gráficas a un formato como pdf, csv... para poder guardar este informe generado y tener así un histórico, o un documento generado que puedes imprimir, transportar o extender de manera offline. Se trata de una función de HTML5 que no requiere de procesos o algoritmos tediosos. Tan sólo con llamar a window.print() se despliega una nueva pestaña con las opciones de exportar, preparado para tener que darle solamente al botón de guardar.

IMPLEMENTACIÓN 17: Visualizar datos de una enfermedad

Dentro de la pestaña de Enfermedades con detalle se encuentra el segundo buscador de la página web. Permite localizar una enfermedad a partir de su nombre y mostrar una serie de datos relevantes para el médico sobre esa enfermedad. Utiliza también la librería *c3.js* para mostrar mediante gráficas la distribución de las personas contagiadas según el sexo y los intervalos de edades. También se calculan gráficas de dispersión relacionando el sexo, la edad y el peso; las tres magnitudes de las que se dispone. De cada enfermedad se muestra el número de contagiados, el número de muertes, si está erradicada y el peso medio al que afecta la enfermedad. Este peso medio se calcula mediante la fórmula de la media aritmética, teniendo un pequeño error pero aproximándose fielmente a la realidad. Gracias a la propiedad de *hidden* y *show,* se puede mostrar un recuadro vacío con el logo del proyecto y, sólo cuando se haya comprobado la entrada del buscador y obtenido los datos de la enfermedad, cambiar la vista y plasmar la información en formato de tablas y gráficas. Todas las llamadas a la API (GET) se recogen en las siguientes urls:

/malarm/api/disease/enfermedad

/malarm/api/dispersion/enfermedad.name

IMPLEMENTACIÓN 18: Generar informe de una enfermedad

Al igual que en su pestaña gemela, la web brinda la posibilidad de plasmar todos los datos y la información obtenida en un pdf obteniendo un documento con el que poder trabajar, un documento que se puede enviar, transportar o difundir sin necesidad de usar la web. Gracias a HTML5 y su función window.print() se puede conseguir el pdf de manera sencilla, rápida y sin necesidad de implementar funciones complicadas.

IMPLEMENTACIÓN 19: Consultar mapa de calor global por enfermedad

Esta funcionalidad se encuentra en la pestaña "*@Social*". En este caso, al pulsar en el botón de buscar, se llama a una función en programada en JavaScript que realiza la consulta a la API Social. Una vez obtenida la respuesta en el formato adecuado, se convierte en objetos de tipo "*LatLng*", que puede entender el mapa proporcionado por la API de *Google Maps* para *JavasScript* y se actualiza en la web.

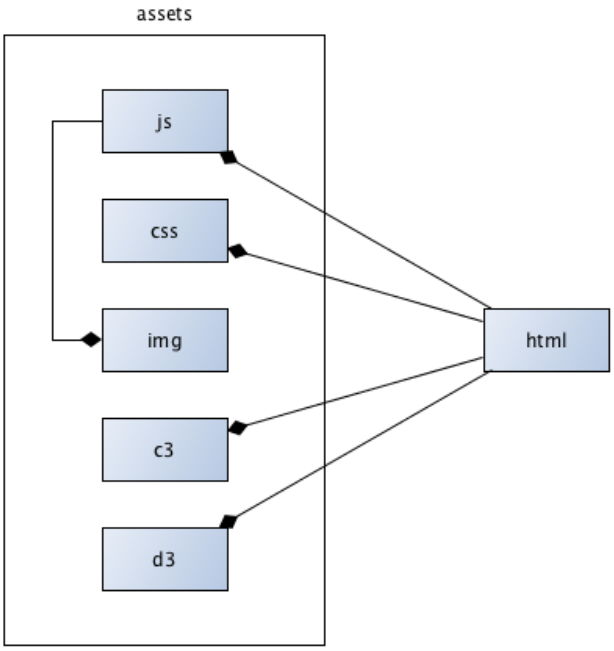
/heatmap?disease=nombre_enfermedad

IMPLEMENTACIÓN 20: Consultar lista de enfermedades globales por zona y fecha

Al igual que la funcionalidad anterior, esta se encuentra en la pestaña "*@Social*". En este caso, al pulsar en el botón "*Buscar*", se llama a una función programada en JavaScript que realiza la conversión de los datos obtenidos como respuesta de la API Social en formato *JSON* y los añade a la lista de enfermedades de la web.

/diseases?zone=zona&date=fecha

A continuación se muestra un esquema, como si de un diagrama de clases se tratase, para aclarar y simplificar la estructura de la página web:



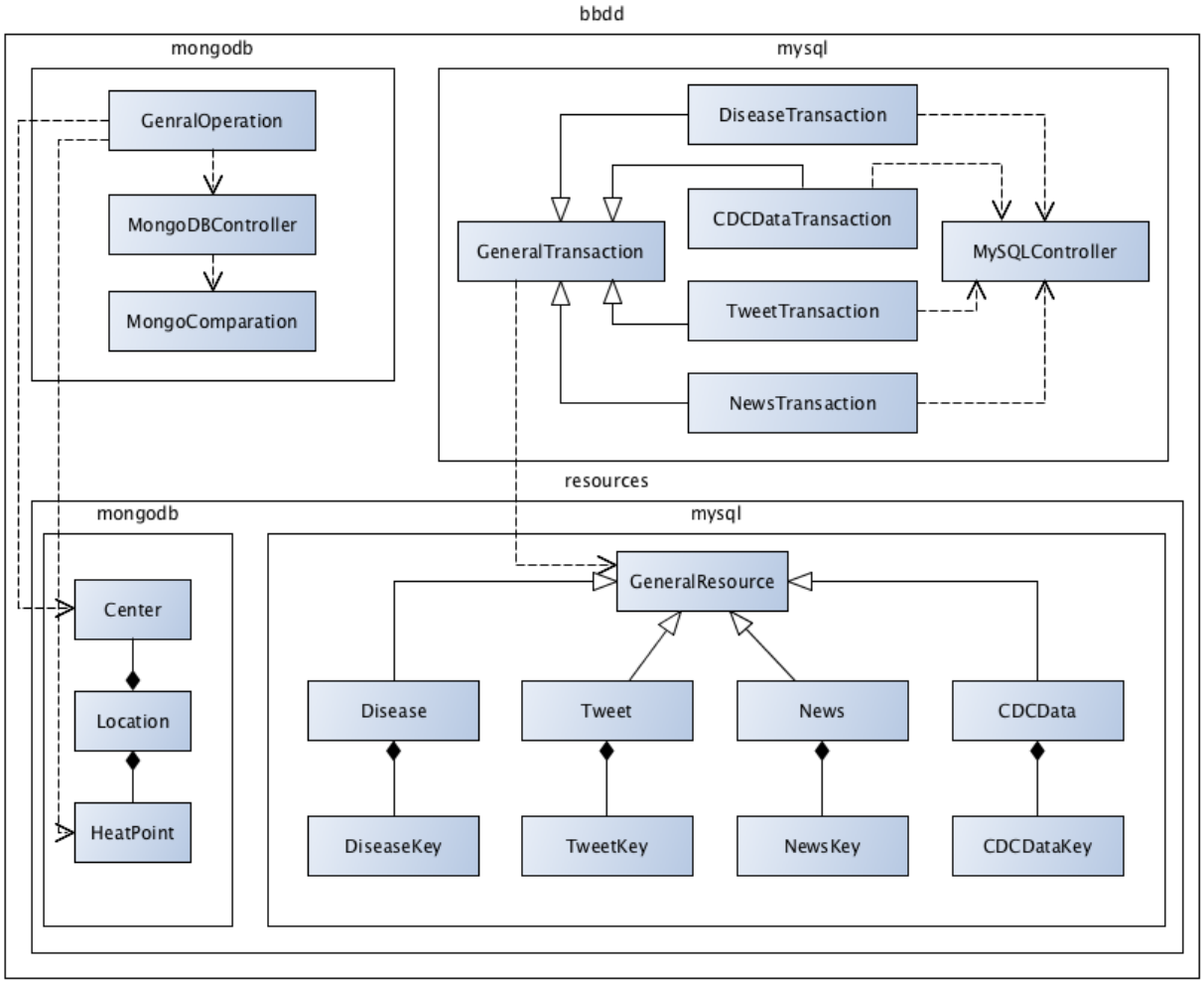
71 - Figura 7.2.3 - Diagrama clases web

7.3 Recolector de datos

El recolector de datos hace uso del framework *Hibernate* para relacionar los objetos con las entidades de la base de datos relacional, utilizando las anotaciones de tablas y columnas. En el caso de que una clave primaria fuese compuesta, estaría representada con su propia clase y anotada como clave embebida en el objeto correspondiente. Además se ha implementado el patrón "*Abstract Mapper*" para generar un acceso abstracto y generalizado a la base de datos. Para el acceso a la base de datos *MySQL*, se configura la conexión mediante la clase *MySQLController* y se realizan transacciones, mediante la clase *GeneralTransacation* que implementa los métodos de insertar, obtener, borrar y actualizar de forma abstracta, haciendo solo falta algunos parámetros específicos como nombre de la tabla, nombre de la tabla o nombre de las columnas. Estos datos se obtienen mediante la extensión de la clase *GeneralTransaction* y la implementación de sus métodos abstractos, además de los objetos genéricos (e.g. Disease).

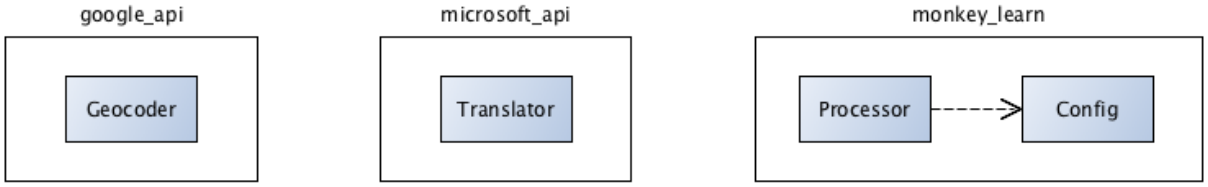
Para realizar consultas y operaciones con la base de datos no relacional en *MongoDB*, se utiliza el framework *Spring Data*. El acceso y la conexión a la base de datos *MongoDB* se configura mediante la clase *MongoDBController* y se realizan las operaciones a través de la clase *GeneralOperation*. En este caso no es necesario extender la clase; para las abstracciones es suficiente con parametrizarla gracias al framework *Spring Data*.

El diagrama de clases de las clases relacionadas con el acceso a bases de datos es el siguiente:



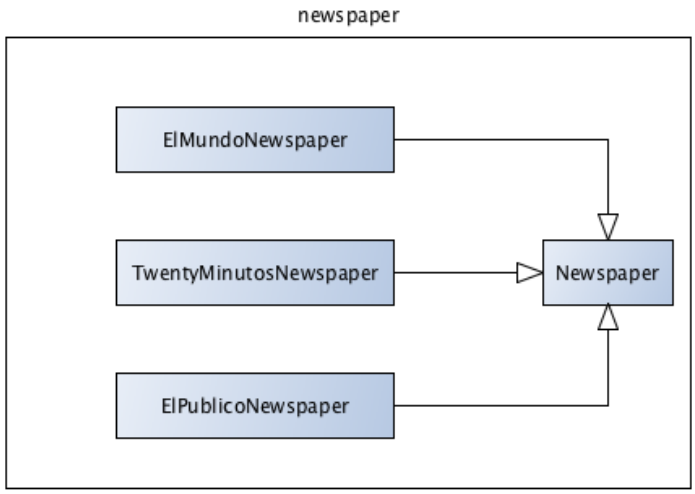
72 - Figura 7.3.1 - Diagrama clases recolector datos

Para el procesamiento y análisis de la información, que será recogida tanto de los periódicos como de Twitter, será necesario el uso de APIs externas. Estas son la API de *Google Geocoding*, *Microsoft Translate* y *MonkeyLearn* (detección de lenguaje, análisis de sentimiento en inglés y español y extracción de entidades en inglés y español). El diagrama de clases del módulo de las APIs es el siguiente:



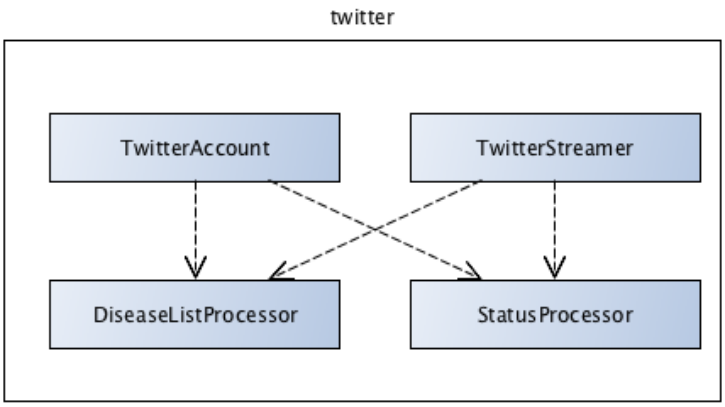
73 - Figura 7.3.2 - Diagrama clases API

El análisis de las noticias de periódicos se ejecuta mediante la clase "*Newspaper*"; esta clase contiene varios métodos abstractos que son implementados por las clases de los periódicos específicos. La parte más importante es la de obtención de noticias; como cada periódico tiene un código *HTML* distinto, esta función se implementa en las clases específicas. Sin embargo, una vez obtenida la lista de noticias, el procesamiento de todas es el mismo. El diagrama de clases de este módulo es el siguiente:



74 - Figura 7.3.3 - Diagrama clases módulo newspaper

El análisis de los datos obtenidos de la red social *Twitter* se ejecuta mediante las clases *TwitterAccounts* y *TwitterStreamer*, que escuchan permanentemente los datos de la red social. En el caso de *TwitterAccounts* se escucha a una serie de cuentas de twitter oficiales que son: *CDCGlobal, sanidadgob, ECDC_Outbreaks, CDCespanol y CDCemergency.* Sin embargo, añadir más cuentas a la lista se puede hacer fácilmente añadiéndolas cuando se invoca el programa. En el caso *TwitterStreamer* se escucha a toda la red social pero filtrando por los nombres de enfermedades de la base de datos (tanto en inglés como en español). El diagrama de clases de este módulo es el siguiente:



75 - Figura 7.3.4 - Diagrama clases módulo twitter

Una vez obtenidos los tweets o las noticas el procesamiento es el mismo, solo cambia la fuente y el peso asignado. El procesamiento es el siguiente:

1. Obtener el lenguaje utilizando la API de *MonkeyLearn*.
2. Según el lenguaje obtener el sentimiento utilizando la API de *MonkeyLearn*.
3. Obtener los nombres de enfermedades mencionados en la noticia, según los existentes en la base de datos.
4. Obtener los nombres de localizaciones mencionados en la noticia, utilizando la API de *MonkeyLearn*.
5. Obtener todas las posibles combinaciones de enfermedad y localización. Si no hubiera ninguna localización, se obtendrían todas las localizaciones de esa enfermedad en la base de datos y se trataría como una noticia genérica.
6. Si no existe una entrada en la base de datos *MySQL* con dicha combinación de enfermedad y localización, se crea; si existe se actualiza su fecha de actualización, se suma una notica o tweet, se actualiza el peso y el nivel de alerta.

7. Si no existe una entrada en la base de datos *MongoDB* con dicha combinación de enfermedad y localización, se crea. Si no existiera un centro en un radio de 500 metros de dicha localización se añade uno.

Además hay otro programa que analiza la sección de alertas de la página del CDC (Centro de Control de Enfermedades). Este programa analiza las entradas, que siempre siguen el mismo formato: nivel, fecha, enfermedad (posible nombre alternativo), lugar (posible especificación). El texto de cada entrada es analizado en busca de los campos anteriores y una vez obtenidos, se consulta la base de datos *MySQL*. En el caso de no existir, se crea la entrada; si existe se actualiza la fecha de modificación y se añade un cdc a la entrada. En este caso no se muestra el diagrama de clases, ya que el módulo consta de una sola clase llamada "*CDCWebNotices*".

La lista de pesos asignados a cada ítem según su fuente, nivel y sentimiento es:

Fuente	Sentimiento	Nivel	Peso
Twitter	Positivo	-	-2
	Neutro	-	2
	Negativo	-	4
Noticias	Positivo	-	-4
	Neutro	-	4
	Negativo	-	8
CDC	-	1	100
	-	2	250
	-	3	500

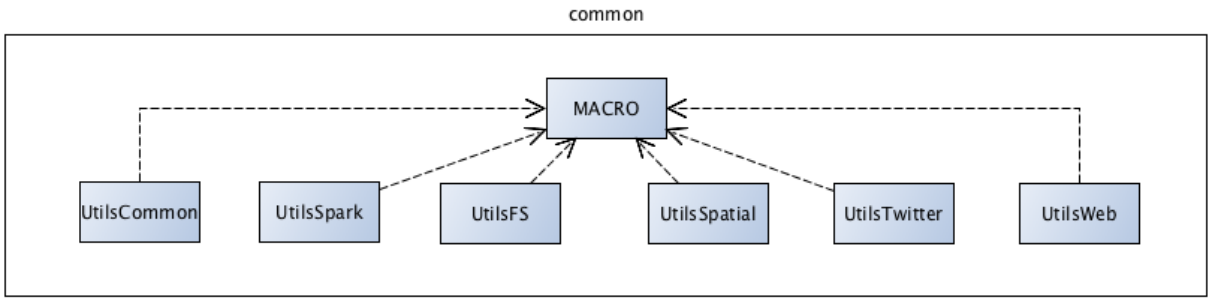
76 - Tabla 7.3.1 - Tabla de pesos

La lista de pesos y niveles generales de la aplicación son:

Peso	Nivel
< 250	1
250 <= peso < 500	2
=> 500	3

77 - Tabla 7.3.2 - Pesos y niveles

Además en muchos casos hay operaciones o constantes comunes a varios módulos, estos están abstraídos en el módulo "*common*". El diagrama de clase de dicho módulo es:



78 - Figura 7.3.5 - Diagrama clases módulo common

7.4 API

Como ya fue explicado en el apartado de funcionalidad, existen dos APIs: una del sistema y otra social.

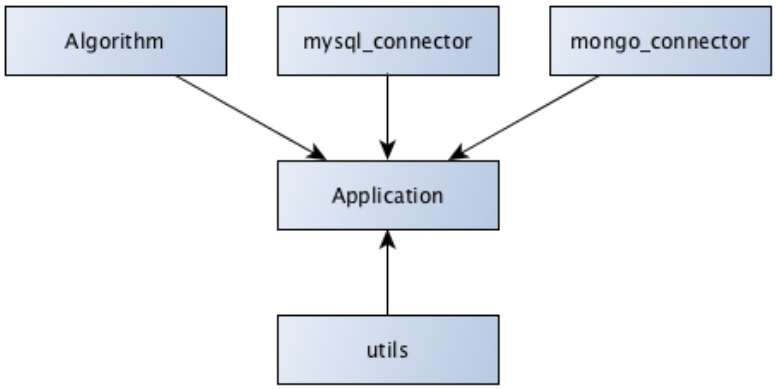
La API del sistema ha sido implementada utilizando *Python 2.7.3* como lenguaje de programación y *Flask* como tecnología de soporte para APIs de tipo RESTful. Para poder conectarse a las bases de datos, se utilizan las librerías *mysqlalchemy* para *MySQL* y *mongoalchemy* para *MongoDB*. Para los cálculos necesarios en el alta de focos y contagios, se utiliza la librería *pyspark* para acceder a spark desde *Python*.

La API social ha sido implementada utilizando *Java 7* como lenguaje de programación y *Spring REST* y *Jersey* como tecnología de soporte para APIs de tipo RESTful. Para poder conectarse a las bases de datos se utilizan los frameworks *Hibernate*, implementando el patrón *Abstract Mapper* para el acceso a *MySQL* y *Spring Data* para el acceso a *MongoDB*.

En ambos casos el lenguaje utilizado para intercambiar datos es JSON.

7.4.1 API del sistema

La API del sistema consta de un fichero *Python* principal, *application.py*, mediante el cual se inicia. Este fichero contiene las clases necesarias para gestionar las distintas peticiones a la API. A su vez, dichas clases hacen uso de otras dos: *mongo_connector* y *mysql_connector*, para realizar las conexiones con las bases de datos y las consultas correspondientes. Además, utilizan la clase *algorithm* cuando es necesario ejecutar algún algoritmo (en el caso de alta de foco o contagio). El diagrama de clases de la API del sistema es el siguiente:



79 - Figura 7.4.1.1 - APi del sistema

La correspondencia entre URLs y clases, y los métodos implementados es la siguiente:

Clase / Recurso	URL	GET	POST	PUT	DELETE
Users	/malarm/api/users/	X	X		
User	/malarm/api/user/<dni>/	X	X	X	
Sensors	/malarm/api/sensors/	X	X		
Diseases	/malarm/api/diseases/	X			
Disease	/malarm/api/disease/<name>/	X			
Focus	/malarm/api/focus/<id>/				X
Focuses	/malarm/api/focuses/	X	X		
Contagion	/malarm/api/contagion/<id>/			X	
Contagions	/malarm/api/contagions/	X	X		
UsersContagions	/malarm/api/users/contagions/	X	X		X
Notifications	/malarm/api/user/notifications/	X			
Notification	/malarm/api/notification/			X	X
Dispersion	/malarm/api/dispersión/<name>/	X			
News	/malarm/api/news/	X			X
Status	/malarm/api/user/status/	X			

80 - Tabla 7.4.1.1 - URLs, clases y métodos 1

Como se puede observar, algunas de las URLs tienen parámetros acotados por los caracteres '<', '>'. Estos parámetros son procesados utilizando expresiones regulares mediante la clase *Utils.* Además, a las URLs hay que añadir la correspondiente dirección IP y puerto del servidor donde se encuentren, siendo el formato de las URLs el siguiente *http://ip:puerto/URL*.

La implementación de los métodos de cada clase, los parámetros de entrada y salida, y su funcionamiento se explican a continuación.

Users

- **GET**: tiene que recibir el parámetro "type" con valor "status". La petición sería la siguiente: http://ip:puerto/malarm/api/users/?type=status. Se llamaría a la función get_total_user_status del objeto mysql_connector que realizaría la consulta a la base de datos. Tras realizar la consulta a la base de datos y el procesamiento de la respuesta de la consulta, la API devuelve la información en JSON con el siguiente formato:

```
{
    "cured": número de personas curadas,
    "dead": número de personas fallecidas,
    "healthy": número de personas sanas,
    "infected": número de personas enfermas,
    "undefined": número de personas indefinidas
}
```

- **POST**: La petición sería la siguiente: http://ip:puerto/malarm/api/users/. No son necesarios parámetros, pero si tiene que recibir los datos de registro de usuario en formato JSON en el cuerpo de la llamada. El formato sería el siguiente:

```
{
    "user":{
        "name": nombre del usuario,
        "lastname": apellidos del usuario,
        "email": correo electrónico del usuario,
        "bithday": fecha de nacimiento del usuario,
        "gender": género del usuario,
        "weight": peso del usuario,
        "idnumber": DNI del usuario,
        "secret": contraseña del usuario
    }
}
```

Se comprobará que todos los campos estén presentes en el cuerpo de la llamada. Si no lo están, ya existe un usuario con el mismo email o DNI, y se devuelve un error de tipo 400. Si no, se llama a la función "create_user" del objeto *mysql_connector* y se inserta el usuario en la base de datos mediante una consulta SQL. Todo nuevo usuario se registra con estado indefinido. Si no hay error de conexión con la base de datos, se devuelve un mensaje sin contenido de tipo 200.

User

- **GET**: La petición se realiza a http://ip:puerto/malarm/api/user/<id>/ . Lo cual significa que <id> sería sustituido por el DNI del usuario. Si no existe ningún usuario con dicho DNI se devuelve un error de tipo 400. Si existe, esta petición devuelve los datos del usuario con ese DNI, junto con los contagios activos que tiene actualmente (cada contagio incluye los datos de la enfermedad a la que pertenece). Mediante el objeto mysql_connector se realiza la consulta: y a partir de ahí se consulta a la tabla "usuarioContagiado" para obtener todos los id de los contagios a los que pertenece el usuario. Para cada uno de esos id se llama a la función "get_contagion" y se rellena la lista. El formato del JSON devuelto es el siguiente:

```
{
    "user":{
                    "name": nombre del usuario,
                    "lastname": apellidos del usuario,
                    "email": correo electrónico del usuario,
                    "bithday": fecha de nacimiento del usuario,
                    "gender": género del usuario,
                    "weight": peso del usuario,
                    "idnumber": DNI del usuario,
                    "secret": contraseña del usuario
                    "contagions":[
                        {
                            "contagion_id": id del contagio,
                            "date": fecha de creación del contagio,
                            "description": descripción del contagio,
                            "disease": {
                                "disease_id": id de la enfermedad,
                                "eradicated": si esta erradicada o no,
                                "name": nombre de la enfermedad,
                                "num_adults": número de adultos afectados,
                                "num_children": número de niños afectados,
                                "num_contagions": número de contagiados total,
                                "num_deaths": número de muertes causadas,
                                "num_elders": número de ancianos afectados,
                                "num_men": número de hombres afectados,
```

```
                                        "num_teenagers": número de jovenes afectados,
                                        "num_women": número de mujeres afectados,
                                        "weight": peso medio de los afectados
                                    },
                                    "disease_id": id de la enfermedad,
                                    "distance": distancia máxima para contagiarse,
                                    "doctor_id": id del doctor,
                                    "level": nivel de alerta,
                                    "place": lugar del contagio,
                                    "time": tiempo de exposición minimo
                                },
                                .
                                .
                                .
                            ]
        }
}
```

- **POST**: La petición se realiza a la misma dirección que la petición GET, pero incluyendo en el cuerpo de esta la contraseña del usuario:

```
{
        "user":{
                "secret": contraseña del usuario
        }
}
```

Esta petición consulta la base de datos a través del objeto mysql_connector y obtiene el usuario mediante una consulta. Si no existe se devuelve un error de tipo 404. Si existe se devuelven los datos del usuario en formato JSON:

```
        {
                "user":{
                        "name": nombre del usuario,
                        "lastname": apellidos del usuario,
                        "email": correo electrónico del usuario,
                        "bithday": fecha de nacimiento del usuario,
                        "gender": género del usuario,
                        "weight": peso del usuario,
                        "idnumber": DNI del usuario,
                        "secret": contraseña del usuario
                }
        }
```

- **PUT**: La petición se realiza a la misma dirección que las peticiones GET y POST, pero añadiendo el nuevo estado en formato JSON en el cuerpo de la petición:

```
{
    "user":{
        "new_status": nuevo estado del usuario
    }
}
```

Solo se acepta un valor de "new_status" a fallecido. Actualizándose mediante el objeto mysql_connector, se actualiza el estado del usuario a 4 (fallecido). A continuación se obtienen todos los contagios a los que pertenecía dicho usuario consultando la tabla "usuarioContagiado", y después para cada contagio se obtiene el id de la enfermedad. Esto se hace para poder actualizar el número de muertes de causadas por cada enfermedad.

Sensors

- **GET**: La petición se realiza a: http://ip:puerto/malarm/api/sensors/?distance=d&latitude=la&longitude=lo . Los tres argumentos de esta petición son obligatorios. Mediante un objeto de la clase *mongo_connector*, se obtienen los puntos que estén a distancia igual o menor que d del punto [la,lo]. El formato del JSON devuelto es:

```
[
    {
        "timestamp": marca de tiempo de cuando se guardo dicha entrada,
        "user_id": id del usuario,
        "location":{
            "type": Point,
            "coordinates":[latitud, longitud]
        }
    }
    .
    .
    .
]
```

- **POST**: La petición se realiza a http://ip:puerto/malarm/api/sensors/ y en el cuerpo de la petición se incluirán los datos a guardar en el siguiente formato JSON:

```
{
    "sensor":{
        "user_dni": dni del usuario,
        "timestamp": marca de tiempo de los datos,
        "latitude": latitud,
        "longitud": longitud,
        "magnetometer": valor del magnetómetro,
        "accelerometer": valor del accelerometro,
        "light": lúmenes,
        "batería": estado de carga/descarga de la batería
    }
}
```

Estos datos son insertados en la base de datos no relacional mediante el objeto mongo_connector.

Diseases

- **GET**: La petición se realiza a http://ip:puerto/malarm/api/diseases/?type=t&top=n . El argumento "type" puede tomar el valor "d" para obtener las enfermedades ordenadas por número de muertes, "c" para obtenerlas ordenadas por número de contagios, o "all" para obtener todas las enfermedades. El parámetro "top" sirve para limitar el número de enfermedades obtenidas. Por ejemplo type=d y top=5 obtendría las 5 enfermedades que más muertes han causado. El formato del JSON devuelto es:

```
[
    {
        "name": nombre de la enfermedad,
        "num_deaths": número de muertes causadas
    },
    .
    .
    .
]
```

NOTA: en el caso de ser type=c en el JSON se sustituiría el campo "num_deaths" por "num_contagions".

Además si type= all se devuelven las enfermedades con todos sus campos:

```
[
        {
                                "disease_id": id de la enfermedad,
                                "eradicated": si esta erradicada o no,
                                "name": nombre de la enfermedad,
                                "num_adults": número de adultos afectados,
                                "num_children": número de niños afectados,
                                "num_contagions": número de contagiados total,
                                "num_deaths": número de muertes causadas,
                                "num_elders": número de ancianos afectados,
                                "num_men": número de hombres afectados,
                                "num_teenagers": número de jovenes afectados,
                                "num_women": número de mujeres afectados,
                                "weight": peso medio de los afectados
          },
                        .
        .
        .
]
```

Disease

- **GET**: La petición se realiza a http://ip:puerto/malarm/api/disease/<name>/ siendo "name" el nombre de la enfermedad. A través de un objeto de tipo mysql_connector se busca la enfermedad en la base de datos y se obtienen su campos, siendo el número de contagiados la suma de hombres y mujeres que padecen o padecieron esa enfermedad. Si la enfermedad no existe, se devuelve un error de tipo 400. Si existe, se devuelven sus datos en formato JSON:

```
{
        "eradicated": si esta erradicada o no,
        "iddisease": id de la enfermedad,
        "name": nombre de la enfermedad,
        "numteenagers": número de jóvenes afectados,
        "numadults": número de adultos afectados,
        "numchildren": número de niños afectados,
        "numcontagions": número de afectados,
        "numdeaths": número de fallecidos,
        "numelders": número de acianos afectados,
        "nummen": número de hombres afectados,
        "numwomen": número de mujeres afectadas,
        "weight": peso medio de los afectados
}
```

Focus

- **DELETE**: La petición se realiza a http://ip:puerto/malarm/api/focus/<id>/ . Siendo id el id del foco. No requiere argumentos pero debe pasarse en el cuerpo de la petición el foco que se quiere borrar, el formato del JSON es el siguiente:

```
{
    "focus_place": {
        "place": lugar del foco
    }
}
```

Si el lugar del foco existe en la tabla "lugaresFoco" se elimina dicha entrada llamando a la función "delete_place_focus" del objeto de tipo mysql_connector. Si no existe el foco se devuelve un error de tipo 404, si se consigue borrar con existo se devuelve un 204 sin contenido de texto.

Focuses

- **GET**: La petición se realiza a http://ip:puerto/malarm/api/focuses/ . Se obtiene la lista de focos y sus lugares. El formato del JSON recibido es el siguiente:

```
[
    {
        "description": descripción del foco
        "doctor_id": id del doctor,
        "focus_id": id del foco,
        "num_people": número de personas en el foco,
        "places": [
            {
                "date": fecha del lugar del foco,
                "focus_id": id del foco,
                "place": lugar del foco
            },
            .
            .
            .
        ]
    },
    .
    .
    .
]
```

- **POST**: La petición se realiza a la misma dirección que la petición GET. Pero se envía en el contenido la lista de DNIs de los usuarios que se quiere obtener los cruces de localizaciones en formato JSON:

```
{
        "focus_users":[
                { "dni": dni del usuario},
                .
                .
                .
        ]
        "description": descripción del foco,
        "doctor_id": id del doctor
}
```

En primer lugar se obtienen los datos de cada usuario mediante un objeto de tipo mysql_connector. A continuación, para cada uno de los usuarios, se obtienen los puntos que hay registrados del usuario y se crea un objeto de tipo "LineString" que identifica el camino de dicho usuario. El siguiente paso es aplicar el algoritmo de focos que será explicado más adelante, para obtener los cruces de caminos de dos o más usuarios.

En el caso de que uno o más focos sean encontrados, estos son insertados en la base de datos mysql. Para ello se hace uso de la API de Google Geocoding, para convertir las coordenadas del foco en una dirección y que pueda ser guardada en formato texto. La petición es la siguiente:

http://maps.googleapis.com/maps/api/geocode/json?latlng=la,lo&sensor=true&key=API_KEY

El formato del JSON de respuesta es:

```
[
        {
                        "num_users": 2,
                        "points": [
                                {
                                        "location":{
                                                "coordinates": [ latitud, longitud],
                                                "type": "Point"
                                        },
                                        "place": dirección del foco
                                },
                                .
                                .
                        ]
        }
]
```

Contagion

- **PUT**: La petición se realiza a http://ip:puerto/malarm/api/contagion/<id>/ . Si existe un contagio con el id especificado se actualiza su estado a inactivo (nivel 0) mediante un objeto de tipo mysql_connector. En caso de error se devuelve un 404, en caso correcto se devuelve un 204 sin contenido.

Contagions

- **GET**: La petición se realiza a http://ip:puerto/malarm/api/contagions/?type=t . El parámetro "type" puede ser "coordinates" o "contagions". En el primer caso se obtiene la lista de puntos donde se encuentran activos los contagios. Para convertir las direcciones en coordenadas se utiliza la API de Google Geocoding con la siguiente petición
  https://maps.googleapis.com/maps/api/geocode/json?address=direccion&key=API_KEY
  El formato del JSON el siguiente:

```
[
    {
                    "lat": latitud,
                    "lng": longitud
    },
    .
    .
    .
]
```

Si "type" tiene el valor "contagions" se obtiene una lista con todos los contagios activos, es decir, el nivel es mayor que 0. El formato del JSON es el siguiente:

```
[
    {
                    "contagion_id": id del contagio,
                    "date": fecha de creación del contagio,
                    "description": descripción del contagio,
                    "disease": {
                            "disease_id": id de la enfermedad,
                            "eradicated": si esta erradicada o no,
                            "name": nombre de la enfermedad,
                            "num_adults": número de adultos afectados,
                            "num_children": número de niños afectados,
                            "num_contagions": número de contagiados total,
                            "num_deaths": número de muertes causadas,
                            "num_elders": número de ancianos afectados,
```

```
                                        "num_men": número de hombres afectados,
                                        "num_teenagers": número de jovenes afectados,
                                        "num_women": número de mujeres afectados,
                                        "weight": peso medio de los afectados
                                },
                                "disease_id": id de la enfermedad,
                                "distance": distancia máxima para contagiarse,
                                "doctor_id": id del doctor,
                                "level": nivel de alerta,
                                "place": lugar del contagio,
                                "time": tiempo de exposición minimo
        },
        .
        .
        .
]
```

- **POST**: La petición se realiza a http://ip:puerto/malarm/api/contagions/ y en el cuerpo de la petición se envían los parámetros necesarios para añadir un nuevo contagio. El formato del JSON es:

```
{
        "user_contagion":{
                                "user_dni": dni del usuario inicial
                                "distance": distancia máxima para el contagio,
                                "time_window": ventana de tiempo hacia el pasado,
                                "exposure": tiempo de exposición minimo,
                                "disease": nombre de la enfermedad,
                                "doctor_id": id del doctor,
                                "date": fecha de inicio,
                                "level": nivel de alerta,
                                "description": descripción
        }
}
```

En primer lugar se comprueba que el usuario inicial exista en la base de datos, si no existe se devuelve un error de tipo 400.
A continuación se ejecuta el algoritmo de contagios que se explicará más adelante. Si se encuentra uno o más posibles contagiados se actualiza la base de datos apropiadamente. Es decir, se añade una notificación a cada posible contagiado. Para procesar la respuesta es necesario de nuevo utilizar la API de Google Geocoding para convertir coordenadas en direcciones. El formato del JSON que se devuelve es:

```
[
        {
                        "date": fecha del contagio,
                        "description": descripción del contagio,g
                        "disease_id": id de la enfermedad,
                        "distance": distancia máxima,
                        "doctor_id": id del doctor,
                        "level": nivel de alerta,
                        "time_window": ventana de tiempo hacia el pasado,
                        "users": [lista de identificadores de los usuarios contagiados],
                        "zone": zona en la que se creó el contagio
        }
  ]
```

UserContagions

- **GET**: La petición se realiza a http://ip:puerto/malarm/api/users/contagions/?disease=d siendo d el nombre de la enfermedad cuyos usuarios contagiados quieren obtenerse. Mediante un objeto del tipo mysql_connector, se recorren las tablas "enfermedad", "contagio", "usuarioContagiado", y "usuario" y se devuelven los usuarios contagiados por la enfermedad dada. El formato del JSON de respuesta es:

```
[
        {
                        "contagion_id": id del contagio,
                        "date": fecha,
                        "description": descripción del contagio,
                        "disease_id": id de la enfermedad,
                        "distance": distancia máxima,
                        "doctor_id": id del doctor,
                        "level": nivel de alerta,
                        "place": zona del contagio,
                        "time": tiempo de exposición minimo,
                        "users": [
                                {
                                        "name": nombre del usuario,
                                        "lastname": apellidos del usuario,
                                        "email": correo electrónico del usuario,
                                        "bithday": fecha de nacimiento del usuario,
                                        "gender": género del usuario,
                                        "weight": peso del usuario,
                                        "idnumber": DNI del usuario,
                                        "state": estado del usuario,
                                }
                        ]
        }
   ]
```

- **POST**: La petición se realiza a http://ip:puerto/malarm/api/users/contagions/ pasando en el cuerpo de la petición el id del usuario que se quiere añadir y el id del contagio al cual se le quiere añadir. El formato del JSON es el siguiente:

```
{
    "user_contagion":{
        "user_id": id del usuario,
        "contagion_id": id del contagio
    }
}
```

  Además se actualiza el número de afectados, peso medio, género, etc. de la enfermedad correspondiente al contagio.

- **DELETE**: La petición se realiza a la misma dirección que la petición POST y el cuerpo de esta es mismo. En este caso se elimina el usuario del contagio y si este no tuviera ningún otro contagio activo se actualiza su estado a 3 (curado). Se devuelve un 204 sin contenido.

Notifications

- **GET**: La petición se realiza a http://malarm/api/user/notifications/?dni=d y se obtienen todas las notificaciones pendientes del usuario con DNI d. En las notificaciones se especifica el contagio y la enfermedad del contagio. El formato del JSON devuelto es:

```
{
    "notifications": [
        {
        "confirmed": "1",
        "contagion": {
                "contagion_id": "1",
                "date": "10/02/2016",
                "description": "lorem impsum",
                "disease": {
                    "disease_id": "1",
                    "eradicated": "no",
                    "name": "Ebola",
                    "num_adults": "15",
                    "num_children": "4",
                    "num_contagions": "31",
                    "num_deaths": "10",
                    "num_elders": "2",
                    "num_men": "15",
                    "num_teenagers": "10",
                    "num_women": "16",
                    "weight": "60"
```

```
                                        },
                                        "disease_id": "1",
                                        "distance": "2",
                                        "doctor_id": "1",
                                        "level": "3",
                                        "place": "Barcelona",
                                        "time": "10"
                                },
                                "contagion_id": "1",
                                "date": "10/02/2016",
                        "user_id": "5"
                        },
                        .
                        .
                        .
                ]
        }
```


Notification

- **PUT**: La petición se realiza a http://ip:puerto/malarm/api/notification/ y se pasa en el contenido de la petición el id del usuario y del contagio que forman la notificación. Mediante un objeto de tipo mysql_connector se actualiza el estado de la notifiación a confirmado. El formato del JSON de la petición es:

```
{
        "notification":{
                "user_id": id del usuario,
                "contagion_id: id del contagio
        }
}
```

  En caso de que no exista una notificación con dicha combinación de id de usuario y contagio se devuelve un error 400, si existe se devuelve un 204 sin contenido.

- **DELETE**: La petición se realiza a la misma dirección que la petición PUT y el contenido de esta es el mismo. Pero en este caso el registro de notificación se elimina de la base de datos mediante un objeto de tipo mysql_connector. Al igual que antes si la combinación de id usuario y contagio no existe se devuelve un error 400, y si existe y se elimina correctamente se devuelve un 204 sin contenido.

Dispersión

- GET: La petición se realiza a http://ip:puerto/malarm/api/dispersion/<name>/ siendo "name" el nombre de la enfermedad de la cual se quieren obtener los datos. Si no existe una enfermedad con dicho nombre en la base de datos se devuelve un error 400. Si existe se devuelve la lista de usuarios con los valores peso, género y edad, siendo esta última calculada como la diferencia de la fecha actual con la fecha de nacimiento registrada en la base de datos. El formato del JSON devuelto es:

```
[
    {
        "weight": peso del usuario,
        "gender": género del usuario,
        "birthday": edad del usuario
    },
    .
    .
    .
]
```

News

- **GET**: La petición se realiza a http://ip:puerto/malarm/api/news/?user_dni=d siendo el DNI del usuario cuyas noticias no leídas se quieren obtener. Si el usuario no existe en la base de datos se devuelve un error 400. Si existe se devuelven, mediante un objeto de tipo mysql_connector, las noticias que el usuario no ha leído (eliminado). El formato del JSON es el siguiente:

```
[
    {
        "contagion": {
            "contagion_id": id del contagio,
                "date": fecha,
                "description": descripción del contagio,
                "disease": {
                    "disease_id": id de la enfermedad,
                    "eradicated": si esta erradicada o no,
                    "name": nombre de la enfermedad,
                    "num_adults": número de adultos afectados,
                    "num_children": número de niños afectados,
                    "num_contagions": número de contagios causados,
                    "num_deaths": número de muertes causadas,
                    "num_elders": número de ancianos afectados,
                    "num_men": número de hombres afectados,
                    "num_teenagers": número de jóvenes afectados,
```

```
                                                    "num_women": número de mujeres afectadas,
                                                    "weight": peso medio de los afectados
                                    },
                                    "disease_id": id de la enfermedad,
                                    "distance": distancia máxima,
                                    "doctor_id": id del doctor,
                                    "level": nivel de alerta,
                                    "place": zona del contagio,
                                    "time": tiempo de exposición
                            },
                            "contagion_id": id del contagio,
                            "description": descripción del contagio,
                            "news_id": id de la noticia
                    },
                                    .
                                    .           .
]
```

- **DELETE**: La petición se realiza a http://ip:puerto/malarm/api/news/ y se pasa en el contenido de la petición el id del usuario y la noticia que se quiere eliminar de la base de datos. El formato del JSON es:

```
{
        "user_news":{
                "user_dni": dni del usuario,
                "news_id": id de la noticia
        }
 }
```

  En el caso de que dicha combinación de ids exista en la base de datos se eliminan mediante un objeto de tipo mysql_connector, devolviendo un 204 sin contenido. En el caso de que no exista se devuelve un erro de tipo 400.

Status

- **GET**: La petición se realiza a http://ip:puerto/user/status/?user_dni=d&lat=la&lng=lo . Se devuelve el estado del usuario (curado, enfermo, etc.) obtenido mediante un objeto de tipo mysql_connector. Y de la zona en la que está para ello se realiza la conversión de coordenadas en lugar utilizando la API de Google Geocoding mencionada anteriormente, y se busca en la base de datos si hay algún contagio con la misma zona en la que está el usuario. El formato del JSON devuelto es:

```
{
	"user": {
		"birthday": fecha de nacimiento
		"contagions": [	{
					"contagion_id": id del contagio,
					"date": fecha del contagio,
					"description": descripción del contagio,
					"disease": {
						"disease_id": id de la enfermedad,
						"eradicated": si esta erradicada o no,
						"name": nombre de la enfermedad,
						"num_adults": número de adultos afectados,
						"num_children": número de niños afectados,
						"num_contagions": número de contagios causados,
						"num_deaths": número de muertes causadas,
						"num_elders": número de ancianos afectados,
						"num_men": número de hombres afectados,
						"num_teenagers": número de jóvenes afectados,
						"num_women": número de mujeres afectadas,
						"weight": peso medio de los afectados
					},
					"disease_id": id de la enfermedad,
					"distance": distancia máxima,
					"doctor_id": id del doctor,
					"level": nivel de alerta,
					"place": zona del contagio,
					"time": tiempo de exposición
				}
		],
		"email": email del usuario,
		"gender": género del usuario,
		"idnumber": dni del usuario,
		"lastname": apellidos del usuario,
		"name": nombre del usuario,
		"state": estado del usuario,
		"user_id": id del usuario,
		"weight": peso del usuario
		},
	"zone": [
				{
					"contagion_id": id del contagio,
					"date": fecha del contagio,
					"description": descripción del contagio,
					"disease_id": id de la enfermedad,
					"distance": distancia máxima,
					"doctor_id": id del doctor,
					"level": nivel de alerta,
					"place": zona del contagio,
					"time": tiempo de exposición
				}
			]
}
```

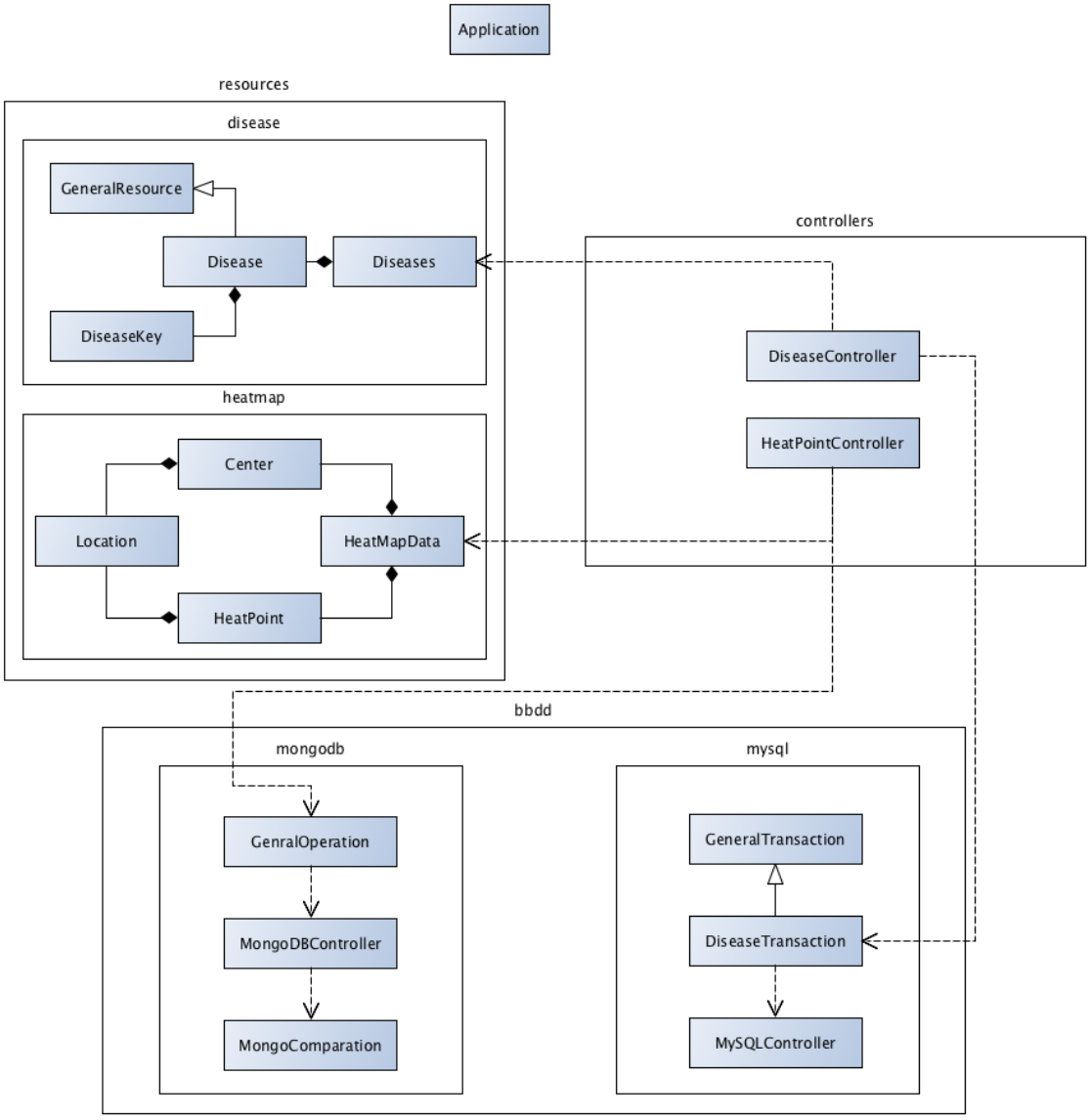
7.4.2 API Social

La API social consta de una clase *Java* principal, *Application.java*, mediante la cual se arranca la API usando el Framework *Spring REST* y *Jersey*. La API controla las peticiones http, mediante las clases controladores.

Para el acceso a la base de datos *MySQL* se configura la conexión mediante la clase *MySQLController* y se realizan transacciones mediante la clase *GeneralTransacation* que implementa los métodos de insertar, obtener, borrar, y actualizar de forma abstracta haciendo solo falta algunos parámetros específicos como nombre de la tabla, nombre de la tabla o nombre de las columnas. Estos datos se obtienen mediante la extensión de la clase *GeneralTransaction* y la implementación de sus métodos abstractos, además los objetos genéricos (e.g. Disease) utilizan las anotaciones del framework *Hibernate* para especificar nombres de tablas y columnas. En el caso de que la clave primaria de alguna tabla fuera compuesta, esta clave se generará en una clase propia y se anotará como clave embebida.

Para el acceso a la base de datos *MongoDB* se configura la conexión mediante la clase *MongoDBController* y se realizan las operaciones a través de la clase *GeneralOperation*. En este caso no es necesario extender la clase ya para las abstracciones es suficiente con parametrizar la clase gracias al framework *Spring Data*.

El diagrama de clases de la API social es el siguiente:



81 - Figura 7.4.2.1 - Diagrama clases API social

La correspondencia entre URLs y clases, y los métodos implementados es la siguiente:

Clase / Recurso	URL	GET	POST	PUT	DELETE
DiseaseController	/diseases	X			
HeatPointController	/heatpoint	X			

82 - Tabla 7.4.2.1 - URL, clases y métodos

DiseaseController

- **GET**: La petición se realiza a http://ip:puerto/diseases y puede llevar el parámetro "zone" y el parámetro "date". El parámetro zone indica la zona en la cual se quiere filtrar las enfermedades (e.g. España), y el parámetro date indica la fecha en la cual se quiere ver las enfermedades activas. Puede realizase una petición con ambos parámetros. Esta petición consulta la base de datos MySQL mediante un objeto del tipo DiseasTransaction. El formato del JSON devuelto es:

```
{
    "diseases":[
                    {
                    "diseaseKey":{
                        "name": nombre de la enfermedad,
                        "location": zona en la que esta activa
                        },
                    "initialDate": fecha de inicio,
                    "lastUpdate": fecha de ultima actualización,
                    "level": nivel de alerta,
                    "weight": peso ,
                    "tweetsCount": número de tweets mencionand la enfermedad,
                    "newsCount": número de noticias mencionando la enfermedad,
                    "cdcCount": número de veces en la web del CDC mentionando la enfermedad,
                    "active": si está activa o no
                    },
                    .
                    .
                    .
    ]
}
```

HeatPointController

- **GET**: La petición se realiza a http://ip:puerto/heatmap y puede llevar el parámetro "disease" para filtrar los por los puntos de una sola enfermedad. Esta petición consulta la base de datos de MongoDB mediante un objeto de tipo GeneralOperation parametrizado con las clases HeatPointList y CenterList. El formato del JSON devuelto es:

```
{
    "heatPointList":[
                    {
                        "weight": peso del punto,
                        "timestamp": marca de tiempo,
                        "name": nombre de la enfermedad,
                        "zone": zona de la enfermedad,
                        "location":{
                            "type":"Point",
                            "coordinates":[latitud, longitud]
                        }
                    },
                    .
                    .
                    .
    ],
    "centerList":[
                    {
                        "name": nombre de la enfermedad,
                        "zone": zona de la enfermedad,
                        "timestamp": marca de tiempo,
                        "location":{
                            "type":"Point",
                            "coordinates":[latitud, longitud]
                        }
                    },
                    .
                    .
                    .
    ]
}
```

7.5 Algoritmos

El sistema hace uso de dos algoritmos:

- El primero se utiliza para el cálculo de contagios. Es decir, dado un usuario inicial, una ventana de tiempo hacia el pasado (cuál es el primer día a tener en cuenta), una distancia máxima y un tiempo mínimo de exposición. El pseudocódigo del algoritmo sería el siguiente:

```
/*
        Siendo user el identificador del usuario inicial, time_window la ventana de tiempo hacia
        el pasado, time_exposure el tiempo de exposición y distance la distancia máxima.
*/
calculateContagion(user, time_window, time_exposure, distance):

found := lista de usuarios contagiados
        uPoints := lista de puntos del usuario user cuya marca de tiempo sea >= marca de
        tiempo actual - time_window

for uPoint in uPoints:

                oPoints := lista de puntos de otros usuarios cuya marca de tiempo sea >=
                marca de tiempo de uPoint - 5 minutos o <= marca de tiempo de uPoint + 5
                minutos

    for oPoint in oPoints:

        if(ditancia(oPoint, uPoint) <= distance)
                        otPoints := lista de puntos del usuario de oPoint en el período
                        de tiempo oPoint + time_exposure
                        utPoints := lista de puntos del usuario uPoint en el período de
                        tiempo uPoint + time exposure

                if(isContagion(otPoints, utPoints))
                        found[n] = oPoint.user
                end if

        end if

    end for

end for

return found;
```

El algoritmo recorre los puntos del usuario en la ventana de tiempo especificada y compara con otros puntos en la misma ventana de tiempo, con un error de +- 5 minutos (ya que ese es el intervalo de tiempo entre el envío de localizaciones si el usuario se mueve), y compara la posición. Si ambos puntos están en una distancia menor especificada se comparan todos sus puntos. Si la condición de distancia se cumple también en todos los puntos en el tiempo de exposición especificado, se añade a dicho usuario a la lista de usuarios contagiados.

- El segundo se utiliza para la búsqueda de los posibles focos de infección dada una lista de usuarios. El pseudocódigo del algoritmo es el siguiente:

```
calculateFocus(users):

                cList[] := contiene todas las posibles combinaciones de usuarios, y cada usuario
                contiene su camino según sus puntos
focus[] := lista de puntos encontrados

for c in cList:

        if((point = foundCross(c)) != null)
                focus[n] = point
        end if
end for

return focus;
```

El algoritmo obtiene los caminos de cada usuario según su lista de puntos (localizaciones). Luego genera todas las posibles combinaciones, es decir, desde los usuarios 2 a 2 hasta todos los de la lista. Una vez generadas las combinaciones se buscan posibles cruces (intersección de las rectas que forman el camino). Si se encuentra se añade a la lista.

Las combinaciones se utilizan debido a que es posible que los usuarios se hayan contagiado en distintos focos, porque lo que una búsqueda de un punto en común a todos los caminos podría dar un falso negativo.

Capítulo 8. Conclusiones y trabajo futuro

8.1 Pruebas y testing

Las pruebas y testing se han realizado con *TestNG* y *JUnit* en el caso de las partes programadas en *Java* y con *PyTest* en el caso de las partes programadas en *Python*.

El acceso a base de datos se ha testeado comprobando las operaciones de insertar, consultar, eliminar y actualizar (en los casos en los que estaban disponibles) para las transacciones en *MySQL* e insertar y consultar para las operaciones en *MongoDB*; mediante tests unitarios realizados con las tecnologías mencionadas anteriormente.

El correcto funcionamiento de las peticiones realizadas a ambas APIs se ha comprobado utilizando el plugin para Google Chrome "*Postman*" y en algunos casos "*Curl*" mediante la terminal de comandos de Linux.

La aplicación móvil ha sido testeada en 4 móviles y actualmente (7 de Junio de 2016) está siendo instalada en más dispositivos. Los dispositivos en los que se conoce su correcto funcionamiento son:

- Motorola Moto G 4G - Android 5.1
- Motorola Moto G – Android 5.1
- Motorola Moto G 3ra Generacion – Android 6.0
- BQ Aquaris E5 – Android 5.0
- Samsung Galaxy S6 – Android 6.0

La aplicación web ha sido testeada en Google Chrome, Mozilla Firefox, Internet Explorer, Spartan. En el caso de Google Chrome y Mozilla Firefox hace falta la instalación del plugin *CORS* para solucionar el problema de "Access-Control Allow Origin". En el caso de Internet Explorer y Spartan la aplicación web funciona sin ser necesaria la instalación de ningún plugin adicional.

El recolector de datos ha sido testeado en un servidor con sistema operativo *OS X El Capitan* y en un servidor con sistema Operativo *Ubuntu 12.04.01*.

Actualmente el sistema se encuentra en funcionamiento en un servidor de la Universidad Complutense de Madrid, en el departamento de Sistemas Informáticos y Computación.

8.2 Conclusiones

Una vez finalizado el proyecto y explicado todo lo recogido en la presente memoria, es un buen momento de echar un vistazo hacia atrás y sacar algunas conclusiones que seguro que serán de gran utilidad a la hora de continuar con la mejora de *MedicalAlarm* o de emprender un nuevo proyecto.

A lo largo de esta etapa de desarrollo, se ha podido comprobar en primera persona que la informática es un sector que se encuentra en continuo avance, que cuando crees que la aplicación Android está terminada, de repente sacan una nueva versión del Sistema Operativo y debes continuar trabajando para actualizarla a los nuevos requisitos; que siempre habrá alternativas en cuanto a las tecnologías, existen un sinfín de lenguajes, de librerías, de módulos, de forma que cuando no encuentres unas librerías de *Django* para la API y se te eche el tiempo encima, decidas pensar fríamente y elegir *Flask* como tecnología para la API.

Éstas son solo algunas de las complicaciones que se han encontrado, seguro que se ha dado alguna más, pero como se puede comprobar, se han sabido solventar. Así que en este sentido se ha conseguido crear una aplicación móvil que avise a los usuarios de posibles contagios, una web para los médicos y especialistas para que administren dichos contagios, y dos algoritmos que forman el grueso del sistema y sin los cuales éste no tendría sentido.

En definitiva, como se mencionó al principio del documento, un proyecto de estas dimensiones, requiere del aprendizaje de multitud de nuevas tecnologías y de estar dispuesto a seguir aprendiendo. Nunca se sabrá todo en informática, es demasiado grande.

8.3 Conclussions

Once the project is finished and everything is expalined in this document, it is a good point to take a look to the past and write down some conclussions that surely will be useful in order to improve *MedicalAlarm* or start another project.

During the development phase it could be noticed in person that computer science is a sector that is in continuous change and when it could have been thought that the mobile application was finished for the Android platform it was found out that a new version was released increasing the workload in order to update and adapt to its requisites. There will always be different technologies to choose from, and when there is a librarie that is not existant for Django and there is no time, the answer is to change to Flask.

These are only some of the complications that have been encountered, it is sure that some have not been mentioned, but as it can be seen they were overcomed. So in this matter the development of a mobile application that alerts users, a web application that helps doctors manage the contagions and two algorithms that form the core of the system without whom the system would make no sense has been achieved.

To sum up, as we mentioned at the beginning of the document, a project of this characteristics requires the learning of several new technologies, and the willing to keep learning. Every aspect of computer science will never be known because it is too broad.

8.4 Trabajo futuro

A continuación se explican algunas de las posibles mejoras que se incluirían en un trabajo futuro:

- **Versión para Android 6**. Como se mencionó en el apartado de conclusiones, hace poco salió una nueva versión de Android, que traía unos nuevos requisitos de permisos que hacían que la aplicación fallase en dispositivos con esa versión. Se consiguió solucionar para que funcionase, pero convendría estudiarlo más en profundidad y crear una versión de la aplicación más estable.

- **Traducciones**. Actualmente, la aplicación está disponible en castellano e inglés, así que otra ampliación futura sería añadir más idiomas para la traducción, ya que se trata de un trabajo poco costoso pero que sí requiere de gente que sepa traducir correctamente los textos usados. Además, se tiene en cuenta también la opción de traducir la página web a dichos idiomas.

- **Envío de localizaciones**. Es algo que no se pudo implementar por falta de tiempo pero que siempre se ha tenido muy en cuenta, y se trata de mejorar el algoritmo de envío de localizaciones, ya que el que existe ahora puede ser demasiado básico y genérico. Por ejemplo, implementando la técnica de *'backoff exponencial'*.

- **Notificaciones**. En este momento la recepción de notificaciones se realiza de manera manual por el usuario, pero la idea es que éstas se reciban en tiempo real a medida que aparecen en el sistema, así que una de las cosas en las que se trabajará en el futuro será un nuevo sistema de notificaciones.

- **Soporte y Ayuda**. Debido a que el ámbito de usuarios de la aplicación es enorme, desde jóvenes hasta no tan jóvenes, el Soporte y la Ayuda de la aplicación puede ser un aspecto vital para el correcto uso de la aplicación, por tanto se ofrecerá un sistema de preguntas frecuentes y un pequeño manual para ayudar a todas estas personas.

- **Integración con @social**. Por último, algo que podría resultar de interés, es poder usar la información que la pestaña @social de la web recopila, para corroborar los datos que el sistema extrae. Por ejemplo, que si una enfermedad provoca un gran número de contagios en poco tiempo, poder validarlo con la repercusión que provoca en las redes sociales y en los periódicos digitales.

8.5 Trabajo individual

8.5.1 Diego Díaz Bailón

Al igual que mis compañeros, he aportado, en mayor o menor medida, en casi todas las partes de este proyecto. Para empezar desde el principio, en la lluvia de ideas inicial todos propusimos ideas que nos resultaban interesantes y las valorábamos en función del trabajo que conllevaría o del uso y la viabilidad que podría tener en un futuro si saliese bien. Después, en las tareas de prototipado, tanto para la aplicación Android como para la web, definimos una serie de prototipos gracias a las opiniones del grupo, mirando siempre por la facilidad de uso. Una vez establecidos los mismos, en mi caso se los enseñé a mi familia para hacerles una prueba de comprensión muy básica, para tener una idea inicial de si la aplicación sería fácil de usar para usuarios de distintas edades y experiencias en tecnologías de este estilo.

Más adelante, cuando tuvimos las ideas claras de lo que queríamos y como lo queríamos, me dediqué a la implementación de la aplicación Android, centrándome sobre todo en la conexión de Android con el *APIRest*. Para ello, inicialmente realicé un curso de la página Udacity para tomar un primer contacto con Android, y posteriormente encontré unos tutoriales que realizaban una tarea parecida a la que queríamos conseguir, así que fueron de gran ayuda y me ayudaron a avanzar enormemente en una tecnología en la que nunca había trabajado. Una vez conseguido esto, nos dividimos las vistas y funcionalidades de la aplicación que ya había empezado mi compañero Manuel, decidimos que yo me encargase de la pestaña de perfil y de la posibilidad de traducir la aplicación, así que me puse a ello.

Cuando obtuvimos una versión más o menos estable, nos centramos en la web. Al igual que con la app, nos repartimos la implementación de las pestañas, decidiéndose en este caso que yo implementaría la de 'Contagios' y la de 'Focos'. No fue difícil pero sí un poco tedioso, ya que la comunicación con mi compañero Pablo fue casi constante puesto que, a pesar de haber pensado y definido previamente las funciones que íbamos a necesitar que implementase en la API, fueron surgiendo nuevas, así que en este aspecto estoy muy contento por el trabajo en equipo que desplegamos.

Así pues, una vez terminada la web "solo" quedaba la implementación de los dos algoritmos principales del sistema: la búsqueda de los focos y la búsqueda de posibles contagios. Nuevamente los tres integrantes aportamos nuestro granito de arena, pero fue Pablo quién llevo a código nuestras ideas, mientras yo volví a Android para acabar los flecos sueltos que faltaban, como la actualización de noticias y datos que nos proporciona la web. En este aspecto acabé la recepción de la actualización del estado del usuario y de la zona (la primera pestaña) y la lista de noticias de la segunda pestaña.

Además, ayudé en el *testing* que realizamos a la aplicación y la instalé en los dispositivos de familiares y amigos para comprobar que funcionase en el mayor número de móviles. Fue un poco costoso "cambiar del chip" y volver de *Javascript* a Android, teniendo que recordar como habíamos implementado ciertas cosas y por qué las habíamos hecho así, para de esta manera poder acabarlas. Pero pensándolo ahora fríamente es una situación que seguramente me volveré a encontrar más adelante y además, felizmente, el resultado ha sido el esperado.

Y por último, la presente memoria, una vez más nos la repartimos equitativamente, dando especial hincapié en que cada uno explicase más en detalle sobre aquellas partes en las que se hubiese involucrado un poco más, como por ejemplo en mi caso, las explicaciones de las funcionalidades y de la implementación de Android o de las partes que realicé de la web.

Como conclusión personal, no pienso que nadie haya hecho más que nadie, es posible que alguno de nosotros haya trabajado más duro en algunas partes, pero luego habrá estado menos agobiado con otras. En este aspecto siempre hemos tenido una buena organización, tanto interna entre nosotros tres, como con nuestro tutor, para tenernos marcadas unas fechas y llevar buen ritmo y no tener que ir corriendo al final para acabarlo a tiempo. Desde el principio supimos que queríamos llevar a cabo un proyecto interesante y que nos gustase, un proyecto con el que disfrutásemos mientras lo realizábamos, y en mi caso así ha sido. Cuando se tiene un buen equipo se trabaja muy a gusto.

83 - Figura 8.5.1.1 - Trabajo Diego

8.5.2 Manuel Martínez Sánchez

Quería empezar dando las gracias a mi dos compañeros por todo el trabajo realizado y recalcar que cada uno de nosotros hemos sido imprescindibles para el desarrollo del proyecto. Todos hemos contribuido de la misma forma, poniendo todo nuestro empeño y esfuerzo en sacar cada parte adelante.

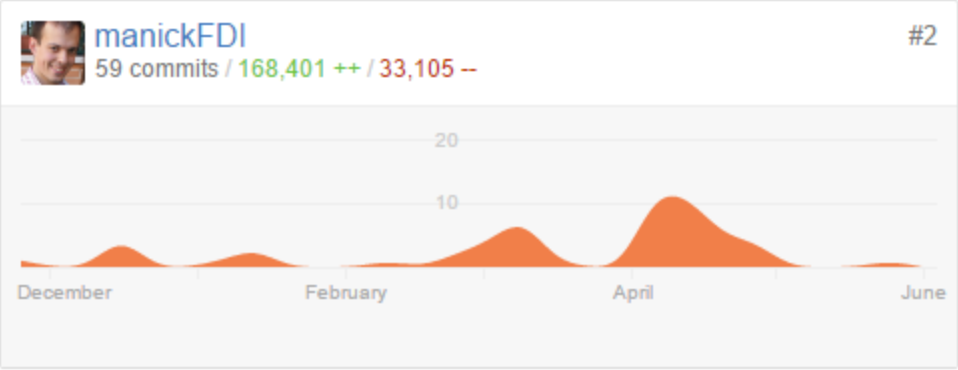
No obstante, cada uno de nosotros tenemos unas cualidades y unas virtudes que hemos explotado para sacar el máximo rendimiento posible.

En mi caso, he colaborado en muchas partes del proyecto y tocado muchas tecnologías y herramientas pero intentaré enumerar y aclarar mi aportación al proyecto brevemente.

Comenzamos el proyecto en octubre y con él, nuestro primer objetivo, aprender Android. En esta primera etapa me dediqué a aprender Android por mi cuenta por medio de tutoriales, libros y cursos online. Al mismo tiempo, dado que a mí me gusta el diseño y había cursado asignaturas relacionadas con ello como interfaces de usuario, me encargué de realizar unos primeros bocetos de las interfaces de la web y de la aplicación móvil. Mediante software de maquetación y prototipado elaboré varios ejemplos de la app y la web. Se los enseñé a mis familiares y amigos y tomé anotaciones acerca de posibles cambios y mejoras. Cuando los bocetos estaban terminados, comenzamos con la implementación de la aplicación Android. Mi trabajo se centró en el desarrollo de la interfaz y dejar todo preparado para que mis compañeros se pelearan con las funcionalidades. Me dediqué a desarrollar el sistema de pestañas, a seguir los patrones de *Material Design* implementando el menú cajón y aprender las buenas prácticas de Android. También ayudé con todo lo relacionado con *"Shared Preferences"*, a guardar el usuario y la contraseña una vez introducidos para no tener que pedirlos cada vez que se abría la aplicación y cualquier cosa que mis compañeros me pidieron. Terminadas las navidades, me dediqué al diseño de la base de datos relacional en *MySQL.* Pensé en un primer momento las tablas que podrían hacer falta, los datos que habría que almacenar y la manera más eficiente de gestionar todos ellos. Cuando tuve el esquema realizado, me reuní con mis compañeros y entre todos terminamos de pulir la base de datos. Hubo que añadir varias tablas, cambiar algunos campos que no tuve en cuenta y reforzar el sistema de referencias y de almacenado de todo lo que se relacionaba con noticias, notificaciones y usuarios. También costó determinar cómo guardar todas las estadísticas de cada contagio y enfermedad pero al final, gracias a mis compañeros, dimos entre todos con la solución óptima. Por otra parte, conseguí una reunión con un médico por medio de una amiga. Afortunadamente, mi amiga estudia medicina en la Universidad Complutense y en sus prácticas en hospital conoció a un médico muy agradable y cercano con el que estableció amistad. Consiguió ponerme en contacto con él y concerté una reunión en el Hospital

Universitario 12 de Octubre. Pude enseñar y exponer nuestro proyecto a un médico del departamento de enfermedades infecciosas. Obtuve ideas del médico, anotaciones y consejos que transmití a mis compañeros y así pudimos entre los tres llevarlo a cabo. Por último, en cuanto a implementación se refiere, el desarrollo de la página web recayó sobre Diego y sobre mí, repartiéndonos a partes iguales el trabajo y teniendo que implementar la pestaña de *Inicio* y *Estadísticas*. El desarrollo de la web fue complicado ya que teníamos que estar en contacto con Pablo, encargado de desarrollar toda la API debido a la interacción que estos dos módulos poseen. Pero no fue complicado con un equipo tan bueno, supimos repartirnos el trabajo, colaborar y ayudarnos en todo. Costó implementar el mapa de calor y añadir los puntos, hacer una interfaz amigable e intuitiva y las gráficas. Quizá este último punto fue el más complicado y en el que invertí más tiempo. Dibujar las gráficas correctamente con los datos que recibía en un JSON, delimitar su posición y definir correctamente sus propiedades no fue tarea fácil pero finalmente pude completarlo siempre con el apoyo y la ayuda de Pablo y Diego. Por último he colaborado en el desarrollo de la memoria, haciendo las partes en las que más he participado llevando a cabo la introducción, los patrones de diseño, la funcionalidad de la web con todos su diagramas, el modelo de datos, la implementación de la web de las dos pestañas que he llevado a cabo y parte del manual de instalación.

Aquí muestro, gracias a *github,* una gráfica con la evolución de mi trabajo:



84 - Figura 8.5.2.1 - Trabajo Manuel

Antes de terminar me gustaría explicar los malos momentos por los que hemos pasado. Al enfrentarnos a tantas tecnologías, hemos tenido períodos en los que hemos estado muy atascados, sin saber avanzar. Un ejemplo de esto fue cuando tuvimos que instalar una API, elegimos *Django* y no éramos capaces de conectar *MongoDB* con *MySQL* y *Django*. Entre todos lo intentamos y también entre todos, decidimos cambiar a *Flask* y conseguir que todo funcione. Fueron momentos complicados al igual que el tener que desarrollar una API por

primera vez. Nunca había desarrollado una y ni siquiera el concepto lo tenía claro pero gracias a mi compañero Pablo conseguí ayudar, avanzar en el proyecto y participar en su desarrollo. Todos hemos pasado por momentos más difíciles, momentos de más estrés o simplemente períodos de desánimo pero hemos sabido ser un grupo, apoyarnos e ir avanzando poco a poco siendo comprensivos y fomentando el buen ambiente ante todo.

Ha sido un placer trabajar al lado de Diego y Pablo; estoy muy orgulloso de haber participado en este grupo y en este proyecto. La complicidad y la unión forjada a lo largo de estos cinco años se han visto reflejadas en este ambicioso trabajo. Sólo con ellos he sido capaz de conseguir este objetivo, MedicalAlarm.

8.5.3 Pablo Panero

Empezamos el proyecto pensando la arquitectura global y con una lluvia de ideas, tecnologías y módulos. Decidimos empezar por la aplicación Android. En mi caso ya había trabajado con anterioridad con aplicaciones móviles y en concreto con los sensores de estos dispositivos. Anteriormente los trabajos realizados eran mediante el framework *AWARE,* por lo que tuve que aprender a acceder y manipular los sensores desde el propio sistema operativo de Android y más tarde encargarme del módulo que se encarga de obtener y actualizar los valores de localización, acelerómetro, etc. Y de la implementación de la funcionalidad que le da la opción al usuario de activar o desactivar dichos sensores. En cuanto a la aplicación Android cabe destacar la gran labor tanto de Manuel como de Diego por la gran interfaz y experiencia del usuario conseguida. La aplicación estaba en estado beta antes de las vacaciones de navidad.

Después de las vacaciones nos pusimos con el diseño de la base de datos relacional sobre la cual recaería la mayor parte de los datos del sistema. Nos reunimos los tres y realizamos algunas modificaciones sobre el diseño que tenía Manuel. Además después de la reunión que tuvo con el Doctor del Hospital 12 de Octubre tuvimos que realizar algunos cambios en el diseño para adaptar algunas funcionalidades e incorporar otras.

Además me encargué de la parte de obtención de datos de la web. Es decir, el procesamiento de los datos del flujo proveniente de twitter, el análisis del código HTML y procesamiento de las secciones de salud de los periódicos elegidos, y las alertas publicadas en la página web del CDC. Además del diseño e implementación de las bases de datos relacional y no relacional que conciernen a este módulo ya que se encuentran separadas. Para el acceso a base de datos tuve que aprender a utilizar el framework *Hibernate* en el caso de *MySQL* y

Spring Data en el caso de *MongoDB.* Además implementé el patrón de acceso a bases de datos "*Abstract Mapper*" para el acceso a la base de datos relacional. Para todo el procesamiento tanto de datos de los periódicos como de Twitter se utiliza *Spark*. Por lo tanto tuve que aprender los conceptos básicos de *Spark* e implementar la funcionalidad necesaria, en algunos casos con con el módulo *Spark Streaming.* Además fue necesaria la implementación de clases que interactuaran con APIs de Internet como: *Google Geocoding*, *Microsoft Translate* y varios módulos de *MonkeyLearn* (análisis de sentimiento y detección de entidades en inglés y español). En los tres casos fue necesario el registro en la página web y la obtención de una clave que identificara unívocamente la aplicación que estaba realizando las peticiones.

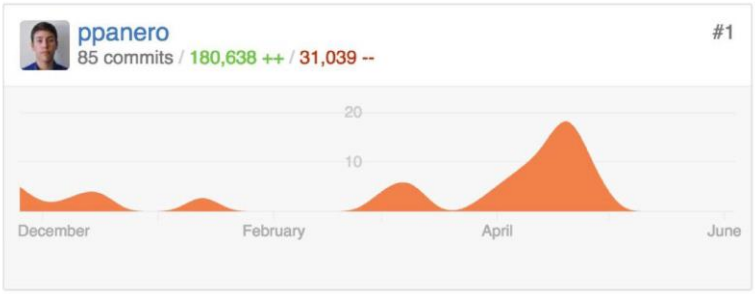
Los datos recogidos desde la web son también accedidos por diversas aplicaciones mediante una API, de cuyo diseño e implementación me encargue yo. Para ello tuve que aprender a utilizar el framework *Spring REST* y el framework *Jersey* ya que ambos se utilizan conjuntamente en este caso.

Una vez realizada la parte de recolección de datos y finalizada la aplicación Android decidimos pasar a la implementación de la página web y la API RESTful. En este caso Manuel y Diego se encargaron de la implementación de la página web y yo de la API. En este apartado la comunicación fue constante ya que pese a haber definido las funcionalidades con anterioridad, los formatos de entrada y salida de datos variaban o incluso surgían nuevas funcionalidades o algunas no eran necesarias. Por mi parte me toco recordar cómo utilizar el framework Flask y el lenguaje de programación Python utilizados para la implementación de la API. Aunque ya había trabajado con ambos en el pasado, no había realizado conexiones a bases de datos ni relacionales ni no relacionales, por lo que tuve aprender a utilizar las librerías *SQLAlchemy* y *MongoAlchemy.* Además yo realice la pestaña "*@Social*" de la aplicación web ya que era la que interactúa con el módulo de recolección de datos mediante la API mencionada anteriormente. Al igual que en el caso de la aplicación Android cabe destacar el trabajo de Diego y Manuel en conseguir una buena interfaz y experiencia de usuario.

En cuanto a la parte de algoritmia, todos pusimos nuestro granito de arena pensando cual era la forma más correcta u óptima de atacar los datos. Una vez pensados los algoritmos y el pseudocódigo entre todos me puse a aprender *PySpark* que es la implementación de *Spark* para el lenguaje de programación *Python* que tenía sus diferencias al usar expresiones "*lambda*", y una vez aprendido implementar los algoritmos pensados anteriormente.

Finalmente una vez acabada la fase de implementación, pasamos a instalar el sistema en uno de los servidores del departamento de Sistemas Informáticos y Computación. Una vez instalado el sistema y dejado en período de prueba pasamos al desarrollo de la memoria repartiéndonos el trabajo según la parte en la que habíamos hecho más esfuerzo durante la fase de implementación.

Por último quiero destacar que hemos sido un "*todos para uno y uno para todos*". Estoy muy contento con el trabajo en equipo y creo que todos hemos trabajado por igual, quizá en mayor o menor medida en algunos módulos pero compensando el balance general. Me alegro mucho de haber compartido esta experiencia y el camino hacia MedicalAlarm con Manuel y con Diego.



85 - Figura 8.5.3.1 - Trabajo Pablo

86 - Figura 8.5.3.2 - Trabajo Pablo (beca de colaboración)

Bibliografía

[1]
«2dsphere Indexes — MongoDB Manual 3.2», *https://github.com/mongodb/docs/blob/master/source/core/2dsphere.txt*. [En línea]. Disponible en: https://docs.mongodb.com/manual/core/2dsphere/.

[2]
«7.3. Métodos Ajax de jQuery (Fundamentos de jQuery)». [En línea]. Disponible en: http://librosweb.es/libro/fundamentos_jquery/capitulo_7/metodos_ajax_de_jquery.html.

[3]
M. Peterman, O. Benomar, H. Mechedou, y F.-H. Bachand, «Address Geocoding Using Street Profiles for Local Search», en *Proceedings of the 25th International Conference Companion on World Wide Web*, Republic and Canton of Geneva, Switzerland, 2016, pp. 655–656.

[4]
«Android >> Integrando un RESTful Web Service en Android |». [En línea]. Disponible en: https://amatellanes.wordpress.com/2014/01/02/android-integrando-un-restful-web-service-en-android/.

[5]
«Android Developers». [En línea]. Disponible en: https://developer.android.com/index.html?hl=es.

[6]
«Apache Spark™ - Lightning-Fast Cluster Computing». [En línea]. Disponible en: http://spark.apache.org/.

[7]
«art%3A10.1186%2F2047-2501-2-3.pdf». .

[8]
«Beautiful Android Login and Signup Screens with Material Design | Sourcey». [En línea]. Disponible en: http://sourcey.com/beautiful-android-login-and-signup-screens-with-material-design/.

[9]
S. I. Hay, D. B. George, C. L. Moyes, y J. S. Brownstein, «Big Data Opportunities for Global Infectious Disease Surveillance», *PLOS Med*, vol. 10, n.º 4, p. e1001413, abr. 2013.

[10]
«Big_Data_Revolution.pdf». [En línea]. Disponible en: http://www.pharmatalents.es/assets/files/Big_Data_Revolution.pdf.

[11]
N. V. Chawla y D. A. Davis, «Bringing Big Data to Personalized Healthcare: A Patient-Centered Framework», *J GEN INTERN MED*, vol. 28, n.º 3, pp. 660-665, jun. 2013.

[12]
H. Chen, R. H. Chiang, y V. C. Storey, «Business Intelligence and Analytics: From Big Data to Big Impact.», *MIS quarterly*, vol. 36, n.º 4, pp. 1165–1188, 2012.

[13]
«C3.js | D3-based reusable chart library». [En línea]. Disponible en: http://c3js.org/.

[14]
«Como obtener la Ubicación actual con el GPS de Android». [En línea]. Disponible en: http://expocodetech.com/expo-tips-como-obtener-la-ubicacion-actual-con-el-gps-de-android/.

[15] «Curso de Programación Android», *sgoliver.net*, 17-jul-2012. .

[16] *Desarrollo de aplicaciones para Android 2016 / Android application development.* Anaya Multimedia-Anaya Interactiva, 2015.

[17] «Flask-PyMongo — Flask-PyMongo 0.4.1 documentation». [En línea]. Disponible en: https://flask-pymongo.readthedocs.io/en/latest/.

[18] «Heatmaps», *Google Developers*. [En línea]. Disponible en: https://developers.google.com/maps/documentation/javascript/examples/layer-heatmap?hl=es.

[19] tutorialspoint.com, «Hibernate Tutorial», *www.tutorialspoint.com*. [En línea]. Disponible en: http://www.tutorialspoint.com/hibernate/.

[20] «HTML Snippets for Twitter Boostrap framework», *Bootsnipp.com*. [En línea]. Disponible en: http://bootsnipp.com.

[21] «Installation — Flask Documentation (0.10)». [En línea]. Disponible en: http://flask.pocoo.org/docs/0.10/installation/.

[22] «monkeylearn», *MonkeyLearn*. [En línea]. Disponible en: http://docs.monkeylearn.com/.

[23] S. Leo y G. Zanetti, «Pydoop: A Python MapReduce and HDFS API for Hadoop», en *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, New York, NY, USA, 2010, pp. 819–825.

[24] tutorialspoint.com, «RESTful Web Services First Application», *www.tutorialspoint.com*. [En línea]. Disponible en: http://www.tutorialspoint.com/restful/restful_first_application.htm.

[25] «Spark Programming Guide - Spark 1.6.1 Documentation». [En línea]. Disponible en: http://spark.apache.org/docs/latest/programming-guide.html.

[26] A. M. Kuchling, «The Python DB-API», *Linux J.*, vol. 1998, n.º 49es, may 1998.

[27] «W3Schools Online Web Tutorials». [En línea]. Disponible en: http://www.w3schools.com/.

[28] «Welcome to Flask-MySQLdb's documentation! — Flask-MySQLdb 0.2.0 documentation». [En línea]. Disponible en: http://flask-mysqldb.readthedocs.io/en/latest/.

[29] «Welcome to Python.org». [En línea]. Disponible en: https://www.python.org/.

[30] «Apps médicas para el sector socio-sanitario», *Inneva Pharma*. [En línea]. Disponible en: http://innevapharma.es/marketing/aplicaciones/.

[31] A. Expósito, «Apps y salud: las apps en el entorno médico», *GoodBarber, Beautiful Apps Engine*. [En línea]. Disponible en: http://blog.goodbarber.com/es/Apps-y-salud-las-apps-en-el-entorno-medico_a310.html.

[32] E. por G.-S. Solutions, «Mejoras en algunos capítulos de Medicina Consultiva», *Go-Space Solutions*, 04-mar-2016. .

[33] «Sanitas en tu dispositivo móvil - Ventajas clientes sanitas». [En línea]. Disponible en: http://www.sanitas.es/sanitas/seguros/es/sobre-sanitas/ventajas-clientes-sanitas/sanitas-en-tu-movil/index.html.

[34] «Una "app" permite controlar en casa a los pacientes con dolor crónico», *ELMUNDO*, 18-feb-2016. [En línea]. Disponible en: http://www.elmundo.es/comunidad-valenciana/2016/02/18/56c5a1324616 3f8e3b8b46a9.html

Apéndices

Apéndice A: Manual de instalación

1. Preparación del entorno

Sistema Operativo

Se recomienda instalar el sistema en un servidor UNIX. Esta guía de instalación tiene en cuenta que se realice sobre un servidor Ubuntu 12.04.4 de 64 bits. Dicho sistema operativo se puede descargar del siguiente enlace:

http://old-releases.ubuntu.com/releases/12.04.0/ubuntu-12.04.4-server-amd64.iso

Al instalar el sistema operativo es conveniente seleccionar "*OpenSSH server*", "*LAMP*" y "*Tomcat Server*" como aplicaciones a instalar, ya que si no tendremos que instalarlas manualmente nosotros más tarde.

Una vez instalado el sistema operativo, si se optó por dicha opción procederemos a instalar los componentes correspondientes a las bases de datos, las APIs y los recolectores de datos web.

NOTA: En el caso de poner comandos a ejecutar en el sistema, el carácter '*$*' hace referencia al *prompt* de la terminal. Además, se debe tener en cuenta que es posible que existan versiones más actuales de algunos de los programas o librerías que vamos a instalar, pero solo se garantiza que el sistema funcione con las especificadas en esta guía.

Primero actualizaremos los paquetes del sistema:

$ sudo apt-get update && apt-get upgrade

$ sudo apt-get install build-essential

Python

Una vez actualizado el sistema, procederemos a instalar Python 2.7.3 y el SDK oficial de Java. Es probable que ambos ya estén instalados en el sistema por defecto, podemos comprobarlo ejecutando los siguientes comandos:

$ python --version

Que debería mostrar "Python 2.7.3" como salida.

$ java –version

Que por defecto mostrará *"java versión 1.6.0_27 Open JDK Runtime Environment…"*, sin embargo el sistema necesita la versión 1.7 de Oracle.

En caso de que Python 2.7.3 no estuviera instalado, podemos hacerlo de la siguiente forma:

$ sudo apt-get install build-essential checkinstall

$ sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev

$ wget http://python.org/ftp/python/2.7.3/Python-2.7.3.tgz

$ tar -xvf Python-2.7.3.tgz

$ cd Python-2.7.3

$ ./configure

$ make

$ sudo checkinstall

Oracle Java 7

El siguiente paso sería instalar el SDK de Java de Oracle. Podemos hacerlo de la siguiente forma:

$ sudo apt-get install python-software-properties

$ sudo apt-add-repository ppa:webupd8team/java

$ sudo apt-get update

$ sudo apt-get install oracle-java7-installer

Es posible que nos pidan aceptar términos y condiciones de uso en ambos casos, tanto para Python como para Java, así pues, si estamos de acuerdo, los aceptamos y seguimos con la instalación.

Una vez instalado Java podemos comprobar que se instaló correctamente verificando la versión:

$ java –versión

Scala

Debería mostrar "*Java versión 1.7.0_80 Java (TM) SE Runtime Environment…"* como salida.

El siguiente paso sería instalar Scala 2.11.7. Lo haremos de la siguiente forma:

$ wget http://downloads.lightbend.com/scala/2.11.7/scala-2.11.7.tgz

$ sudo mkdir -p /usr/local/src/scala

$ sudo tar xvf scala-2.11.7.tgz -C /usr/local/src/scala/

Debemos añadir dos variables de entorno en el fichero *bashrc* que se encuentra en el directorio raíz del usuario, para ello lo abriremos con un editor de texto, por ejemplo *nano*:

$ cd ~ && sudo nano .bashrc

Y añadiremos las dos siguientes líneas:

export SCALA_HOME=/usr/local/src/scala/scala-2.11.7

export PATH=$SCALA_HOME/bin:$PATH

Actualizamos *bashrc* ejecutando el siguiente comando:

$. ~/.bashrc

Comprobamos la versión de Scala:

$ scala –version

Que debería mostrar "*Scala code runner versión 2.11.7…"* como salida.

Apache Spark

El siguiente paso es instalar Apache Spark, para el cual necesitaremos instalar también git. Lo haremos de la siguiente forma:

$ sudo apt-get install git

$ wget http://d3kbcqa49mib13.cloudfront.net/spark-1.5.2.tgz

$ tar xzvf spark-1.5.2.tgz.gz

$ cd spark-1.5.2

$ sbt/sbt assembly

Moveremos la carpeta de archivos de spark a un lugar más apropiado y daremos permisos de propietario al usuario local.

$ mv spark-1.5.2 /usr/local/src/

$ chwon –R <nombre__de_usuario> /usr/local/src/spark-1.5.2

Al igual que antes, debemos añadir dos variables de entorno en el fichero *bashrc* que se encuentra en el directorio raíz del usuario, para ello lo abriremos con un editor de texto, por ejemplo *nano*:

$ cd ~ && sudo nano .bashrc

Y añadiremos las dos siguientes líneas:

export SPARK_HOME=/usr/local/src/spark-1.5.2

export PATH=$PATH:$SPARK_HOME/bin

Actualizamos *bashrc* ejecutando el siguiente comando:

$. ~/.bashrc

MySQL

Una vez instaladas las librerías necesarias, procederemos a instalar los gestores de bases de datos. Si no instalamos MySQL a la par que el sistema operativo, lo haremos ahora ejecutando:

$ sudo apt-get install mysql-server

Podemos comprobar la versión con:

$ mysql --version

El sistema ha sido probado y funciona con versiones 5.6.26 y 5.5.49

Nos pedirá que introduzcamos la contraseña para el usuario *root*. Lo hacemos y esperamos a que termine la instalación.

MongoDB

También es necesario instalar MongoDB, lo haremos ejecutando los siguientes comandos:

$ sudo apt-get install mongodb

Podemos comprobar la versión de MongoDB podemos ejecutar:

$ mongo –versión

El sistema ha sido probado y funciona para las versiones 3.2.0 y 2.0.4

2. Bases de Datos y usuarios del sistema

MySQL

En la carpeta de instalación encontraremos una carpeta llamada "bases_de_datos", en ella se encuentra el archivo "*malarm_mysql_schema.sql*" que deberemos ejecutar. Para ello ejecutamos:

$ cd instalación/bases_de_datos/

$ mysql –u root –p < malarm_mysql_schema.sql

Nos pedirá la contraseña de root, la introducimos. Además podremos comprobar que las tablas se han creado correctamente.

NOTA: Si la base de datos no existiese, el sistema es capaz de crearla al arrancar por primera vez. Sin embargo se aconseja que se cree con anterioridad para evitar y detectar con mayor facilidad posibles problemas. Además, si se desea, se podría ejecutar el script "*malarm_mysql_population.sql*" de la misma forma que el anterior para poblar la base de datos con archivos de prueba.

MongoDB

En este caso solo será necesario crear el usuario y contraseña, para ello ejecutaremos el script "*mongodb_user_creation.json*" que se encuentra en la carpeta "*bases_de_datos*". Tanto la base de datos como las colecciones se crean en tiempo de ejecución según el sistema las va necesitando, por lo que no las crearemos con anterioridad.

Para crear el usuario utilizaremos el comando correspondiente a la versión que esté instalada de MongoDB:

$ mongo < mongodb_user_creation_v2_0_4.json

ó

$ mongo < mongodb_user_creation_v3_2_4.json

NOTA: Antes de ejecutar el comando anterior se deberá haber arrancado el servicio de mongodb. Esto se puede hacer de la siguiente manera:

$ sudo service mongod start

Además, si se desea, se podría ejecutar el script "*mongodb_population.json*" con el siguiente comando:

$ mongoimport --db malarm --collection sensors --file mongodb_population.json

2. Iniciando el sistema

API del Sistema

El primer paso es arrancar la API del sistema. Daremos por hecho que se han arrancado tanto mysql como mongodb, ya que deberían estar arrancados de antes. En caso contrario, ejecutar:

$ sudo service mysqld start

$ sudo service mongod start

Para arrancar la API del sistema deberemos ejecutar el fichero "*application.py*", que se encuentra en *instalación/apis/api_del_sistema/application/.* Es posible que haya paquetes de python que se necesiten, para instalarlos es posible que haya que instalar *pip* primero:

$ sudo apt-get install python-pip

Y a continuación los paquetes que vayan siendo necesarios, en nuestro caso fueron los siguientes:

$ sudo pip install flask

$ sudo pip install falsk-restful

$ sudo pip install py4j

$ sudo pip install --upgrade shapely

Es posible que nos pida instalar GEOS v3.3 o mayor antes de continuar. Podemos instalarlo de la siguiente forma:

$ wget http://download.osgeo.org/geos/geos-3.5.0.tar.bz2

$ tar xfj geos-3.5.0.tar.bz2

$ cd geos-3.5.0

$ ./configure

$ make

$ sudo make install

$ sudo apt-get install libgeos-dev

Después ejecutamos de nuevo el comando "sudo pip install --upgrade shapely" y proseguimos:

$ sudo pip install pymongo==2.7.2

$ sudo apt-get install python-mysqldb

$ sudo pip install requests

$ sudo pip install SQLAlchemy

Finalmente para ejecutar la API del sistema ejecutaremos:

$ cd apis/api_del_sistema/

$ nohup python application/application.py & > api_sistema.log

Página Web

En primer lugar copiaremos la carpeta "*pagina_web*" y todo su contenido a un directorio más apropiado:

$ sudo mkdir /usr/share/malarm_web && sudo cp –r instalación/pagina_web /usr/share/malarm_web

Después crearemos el archivo de despliegue en el servidor Tomcat:

$ nano /etc/tomcat6/Catalina/localhost/malarm_web.xml

Y copiamos el siguiente texto:

<Context path="/malarm_web" docBase="/usr/share/malarm_web/"/>

Suponiendo que el servidor tomcat esté lanzado podremos comprobar que la web está activa accediendo a:

http://localhost:8080/malam_web

http://147.96.80.89:3389/malarm_web

3. Recolector de datos sociales, base de datos y API

MySQL

En la carpeta de instalación encontraremos una carpeta llamada "bases_de_datos", en ella se encuentra el archivo "bcolab_*mysql_schema.sql*" que deberemos ejecutar. Para ello ejecutamos:

$ cd instalación/bases_de_datos/

$ mysql –u root –p < bcolab_mysql_schema.sql

Nos pedirá la contraseña de root, la introducimos. Además podremos comprobar que las tablas se han creado correctamente.

NOTA: Si la base de datos no existiese, el sistema es capaz de crearla al arrancar por primera vez. Sin embargo se aconseja que se cree con anterioridad para evitar y detectar con mayor facilidad posibles problemas. Además, si se desea, se podría ejecutar el script "*bcolab_mysql_population.sql*" de la misma forma que el anterior para poblar la base de datos con archivos de prueba.

MongoDB

En este caso solo será necesario crear el usuario y contraseña, para ello ejecutaremos el script "*mongodb_user_creation.json*" que se encuentra en la carpeta "*bases_de_datos*". Tanto la base de datos como las colecciones se crean en tiempo de ejecución según el sistema las va necesitando, por lo que no las crearemos con anterioridad.

Para crear el usuario utilizaremos el comando correspondiente a la versión que esté instalada de MongoDB:

$ mongo < mongodb_user_creation_v2_0_4.json

ó

$ mongo < mongodb_user_creation_v3_2_4.json

NOTA: Antes de ejecutar el comando anterior se deberá haber arrancado el servicio de mongodb. Esto se puede hacer de la siguiente manera:

$ sudo service mongod start

Además, si se desea, se poblar la base de datos con los siguientes comandos:

$ mongoimport --db diseasemeter --collection centers –file bcolab_centers_mongodb_population.json

$ mongoimport --db diseasemeter --collection centers –file bcolab_heatpoints_mongodb_population.json

API Social

Para iniciar la API Social hay que ir a la carpeta "*api_social*" y ejecutar el siguiente comando:

$ nohup java –jar apis/api_social/ restful_api-1.0-SNAPSHOT.jar *& > api_social.log*

Recolector de datos

Para poner en funcionamiento el recolector de datos es necesario iniciar dos aplicaciones en segundo plano y añadir 4 procesos al cron del sistema. Todos los archivos necesarios se encuentran en "*recolector_de_datos*".

Las aplicaciones en segundo plano se inician mediante los siguientes comandos:

$ nohup spark-submit --class com.diseasemeter.data_colector.twitter.TwitterStreamer --master "local[2]" data_colector-1.0-SNAPSHOT-jar-with-dependencies.jar -m local[2] -t 1000 & > twitter_streamer.log

$ nohup spark-submit --class com.diseasemeter.data_colector.twitter.TwitterAccounts --master "local[2]" data_colector-1.0-SNAPSHOT-jar-with-dependencies.jar -m local[2] -u "CDCemergency, CDCGlobal, CDCespanol, ECDC, ECDC_Outbreaks, sanidadgob& > twitter_account.log

Para añadir los procesos al cron del sistema hay que acceder al editor de cron del sistema, se puede hacer ejecutando:

$ crontab –e

Acto seguido hay que añadir 4 entreadas al fichero que son las siguientes (tener en cuenta que <ruta_absoluta> debe ser sustituida por la ruta donde se encuentran los ejecutables):

0 12 * * * *java -cp <ruta_absoluta>/data_colector-1.0-SNAPSHOT-jar-with-dependencies.jar com.diseasemeter.data_colector.cdc.CDCWebNotices -u* http://wwwnc.cdc.gov/travel/notices

20 12 * * * *spark-submit --class com.diseasemeter.data_colector.newspaper.TwentyMinutosNewspaper --master "local[2]" data_colector-1.0-SNAPSHOT-jar-with-dependencies.jar -u* http://www.20minutos.es/salud/

40 12 * * * *spark-submit --class com.diseasemeter.data_colector.newspaper.ElPublicoNewspaper --master "local[2]" data_colector-1.0-SNAPSHOT-jar-with-dependencies.jar -u* http://www.publico.es/sociedad/sanidad

0 13 * * * *spark-submit --class com.diseasemeter.data_colector.newspaper. ElMundoNewspaper --master "local[2]" data_colector-1.0-SNAPSHOT-jar-with-dependencies.jar -u* http://www.elmundo.es/salud.html

4. Aplicación Android

A continuación debemos instalar la aplicación móvil en los dispositivos necesarios. Para instalar la aplicación en un dispositivo hay dos posibilidades.

Instalación utilizando adb

La primera es utilizar ADB (Android Debug Bridge). Para ello hay que abrir un terminal y seguir estos pasos:

```
$ adb devices
```

Se debería obtener una lista de los dispositivos conectados, dónde se vea nuestro móvil conectado. A continuación habría que navegar hasta dónde se encuentran los ficheros "*.apk*", por defecto están en "*Instalacion/android/":*

```
$ cd .../Instalacion/android/
```

Y a continuación instalar la aplicación correspondiente. Hay dos versiones, una para Android 6.0 o superior y la otra para versiones anteriores:

```
$ adb install medicalAlarm_android6.apk
```

```
$ adb install medicalAlarm.apk
```

Instalación utilizando el explorador de archivos del dispositivo móvil

Primero es necesario habilitar la opción de orígenes desconocidos del menú Ajustes. Eso se encuentra en:

Ajustes->Seguridad

Una vez obtenido el fichero .apk apropiado (hay dos versiones, una para Android 6.0 o superior y la otra para versiones anteriores) en el dispositivo. El siguiente paso es iniciarlo y aceptar los términos y condiciones de uso.

Apéndice B: Manual de usuario

1. Aplicación Android

El objetivo de este manual es acercar y facilitar al usuario el uso de nuestra aplicación. Para ello lo dividiremos por las principales funcionalidades relacionadas con la vista en la que se encuentran, de esta manera el usuario sabrá qué se puede hacer en cada una de las vistas de la app.

1- Login

El login es la primera pantalla que se le muestra a los usuarios la primera vez que usan la aplicación. Para logearse lo único que hay que hacer es rellenar el campo de DNI (1) y de contraseña (2) y pulsar el botón Entrar (3). En caso de ser correcto se pasará a la pestaña 'Estado' (ir al punto 3 del manual).

Figura B.1.1 - Manual: Login

2- Crear cuenta

Si pulsamos sobre el botón Crear cuenta (4 de la imagen anterior) se nos aparecerá la pantalla de registrarnos creando una nueva cuenta. Para ello simplemente debemos rellenar los campos solicitados y confirmar. En cuanto el registro finalice nos aparecerá la vista de 'Estado' (ir al punto 3 del manual).

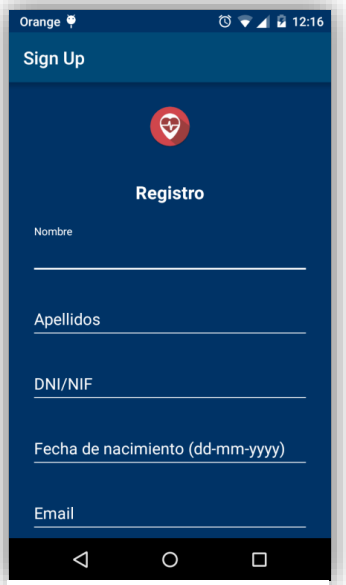


Figura B.1.2 - Manual: registro

3- Pestaña Estado

En esta pestaña podremos ver nuestro estado actual (1) y el de nuestra zona (2). Por ejemplo, en la imagen podemos ver que el usuario no tiene riesgo de estar contagiado, pero en la zona en la que se encuentra sí existe algún contagio. Para actualizar la información es suficiente con realizar el gesto swipe (3).

Figura B.1.3 - Manual: Estado actual

Figura B.1.4 - Manual: Estado actual (sw ipe)

4- Pestaña Noticias

En esta pestaña aparecerán las noticias que aún no hayamos leído o eliminado. Se trata de noticias comunes para todos los usuarios e informan sobre nuevos contagios o sobre la extinción de los mismos.

Para eliminar una noticia es tan simple como pulsar en la X situada en la parte inferior derecha de cada una (4).

De nuevo podemos hacer el gesto de swipe para recargar las noticias.

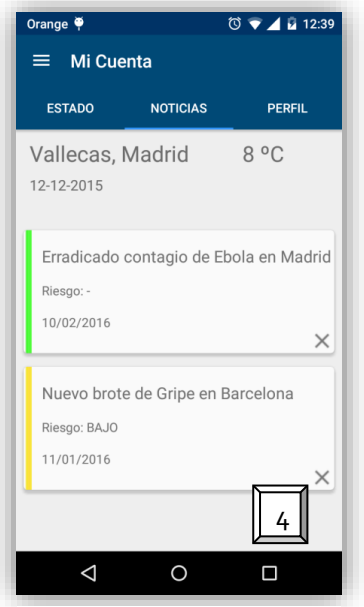


Figura B.1.5 - Manual: Noticias

5- Pestaña Perfil

Se trata de una vista puramente informativa en la que podremos los datos personales que introducimos en el registro. Más adelante se implementará la actualización/modificación de los mismos.

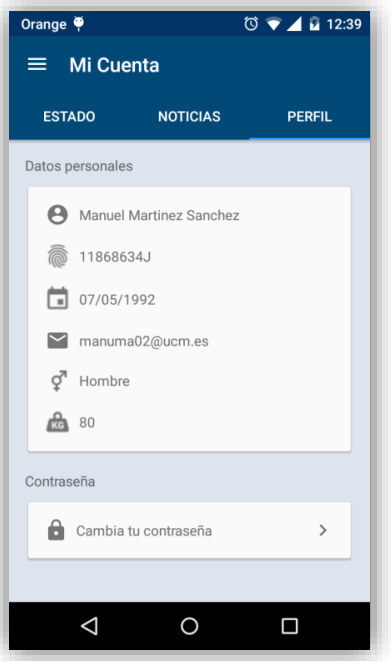


Figura B.1.6 - Manual: Perfil

6- Ajustes

Si pulsamos el icono superior izquierdo de cualquier vista (5) se nos abrirá el menú lateral de la aplicación (6), en el cual nos aparecerán otras opciones de la aplicación, la que nos interesa ahora es Ajustes (7). A continuación se explica cada uno de ellos en detalle.

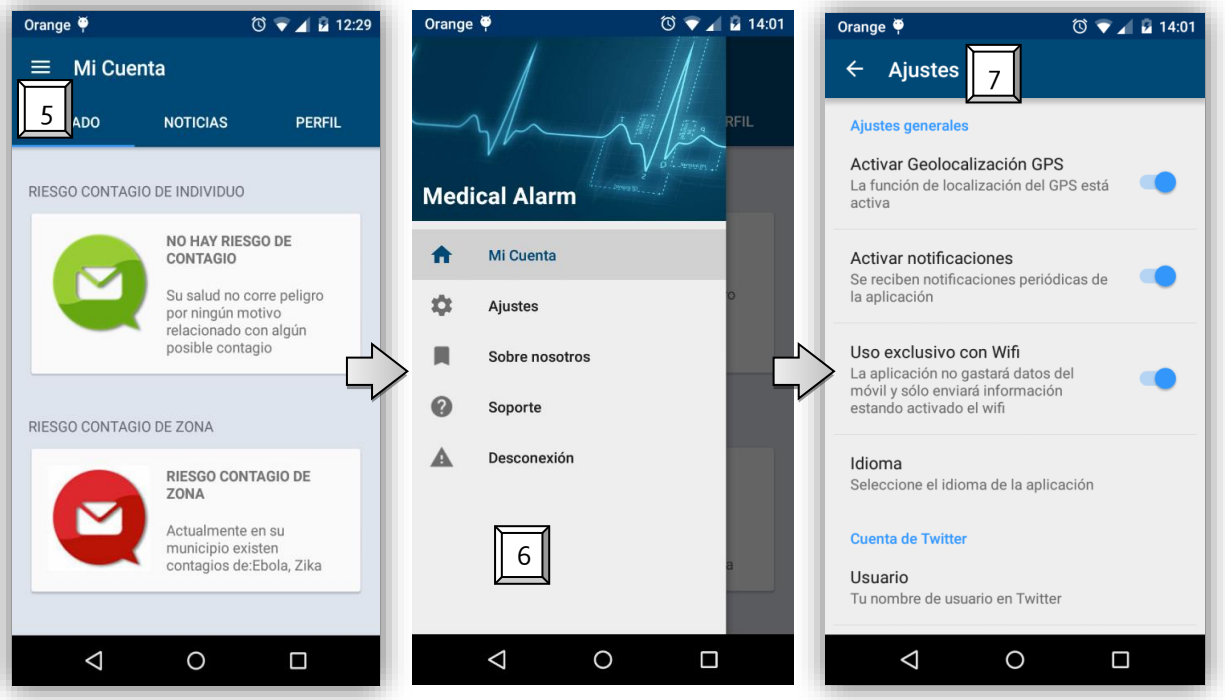


Figura B.1.7 - Manual: Ajustes

7- Cambiar idioma

Si en la vista de Ajustes presionamos sobre la opción Idioma (8), nos aparecerá una nueva vista en la que podremos seleccionar uno de los idiomas que estén disponibles en ese momento para traducir la interfaz (9). Inicialmente la aplicación aparecerá en el idioma por defecto del dispositivo.

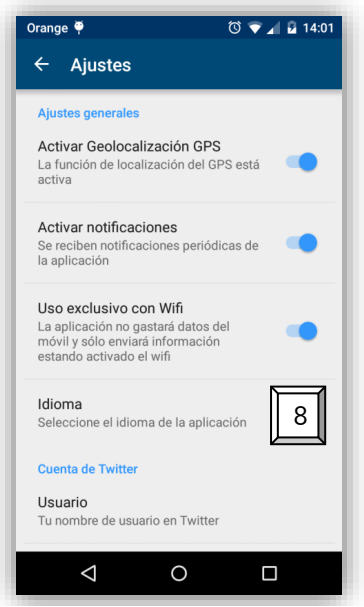


Figura B.1.8 - Manual: Cambiar idioma

Figura B.1.9 - Manual: Seleccionar idioma

8- Activar/desactivar envío de datos de los sensores

Nuevamente en el menú de Ajustes, nos aparece una opción llamada Activar Geolocalización GPS (10) junto a un interruptor o *'switch'* mediante el cual poder activar o desactivar el envío de datos de los sensores (11).

Figura B.1.10 - Manual: Activar/desactivar sensores

2. Página web

En este apartado se muestran las instrucciones necesarias para poder llevar a cabo todas las acciones que la web ofrece.

Figura B.2.1 - Manual: Mapa

1 – Para cambiar la vista del mapa para una mejor visualización pulsar sobre Mapa o Satélite.

2 – Para acercar el mapa y con él, los puntos de calor (contagios), hacer scroll con la rueda del ratón hasta que se desee.

3 – Para cambiar la vista del mapa y abrir Street View para tener una imagen real del terreno, mantener pulsado el icono de la persona naranja y arrastrar al mapa al lugar deseado

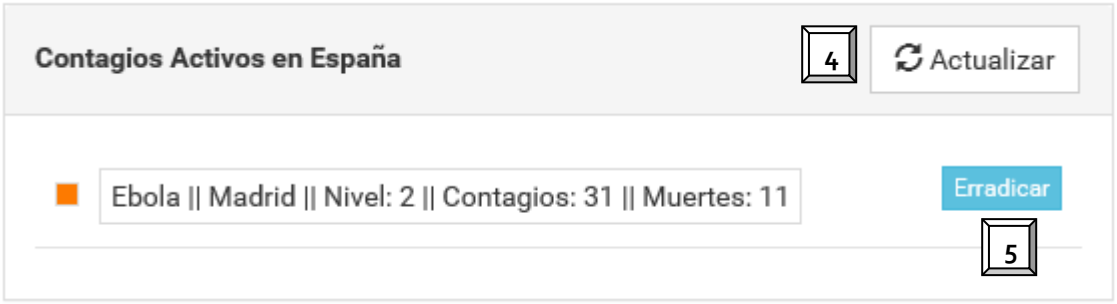


Figura B.2.2 - Manual: Leyenda mapa 1

4 – Para refrescar la leyenda del mapa por si se han producido cambios durante un período de inactividad, pulsar el botón de Actualizar

5 – Para eliminar un contagio activo del mapa de calor, de la leyenda y con ello poner el nivel del contagio a 0, pulsar el botón de Erradicar

Figura B.2.3 - Manual: Buscar usuario

6 – Para buscar a un paciente dentro del sistema, introducir su DNI completo y sin espacios en el buscador y pulsar sobre el icono de la lupa para ejecutar la búsqueda.El usuario en caso de haber sido encontrado, se muestra en el recuadro inferior

NOTA: si el DNI es incorrecto se muestra un mensaje de aviso al igual que si no se ha encontrado el usuario con ese DNI especificado

Figura B.2.4 - Manual: Perfil usuario

7 – Para hacer desaparecer una enfermedad de un paciente y que se elimine de su lista de enfermedades, pulsar sobre el botón Curar

NOTA: Si el paciente se queda sin enfermedades, su estado se actualizará automáticamente a curado

8 – Para marcar a un paciente como fallecido y cambiar su estado, pulsar sobre el botón Usuario fallecido

ATENCIÓN: el botón de Enviar mensaje no está implementado por el momento. Al pulsarlo no se envía ningún mensaje pero en posibles actualizaciones se incluirá.

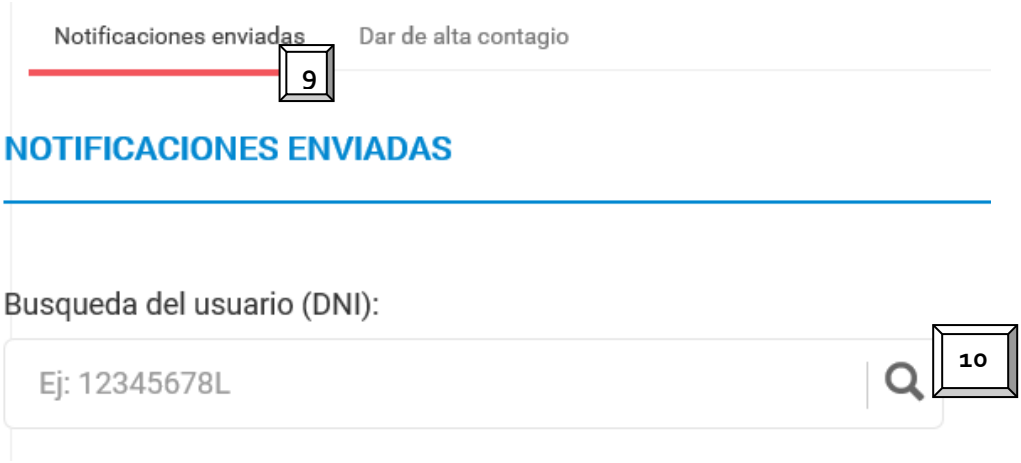


Figura B.2.5 - Manual: Buscar notificaciones

9 – Dentro de la pestaña de contagios, pulsar sobre la pestaña que se quiera en función de lo que se necesite; o visualizar notificaciones enviadas o dar de alta un contagio

10 – Para buscar las notificaciones enviadas a un determinado usuario, introducir en el buscador el DNI completo sin espacios y hacer click sobre el icono de la lupa

NOTA: en caso de introducir un DNI incorrecto o de no ser encontrado, se mostrará un mensaje de aviso

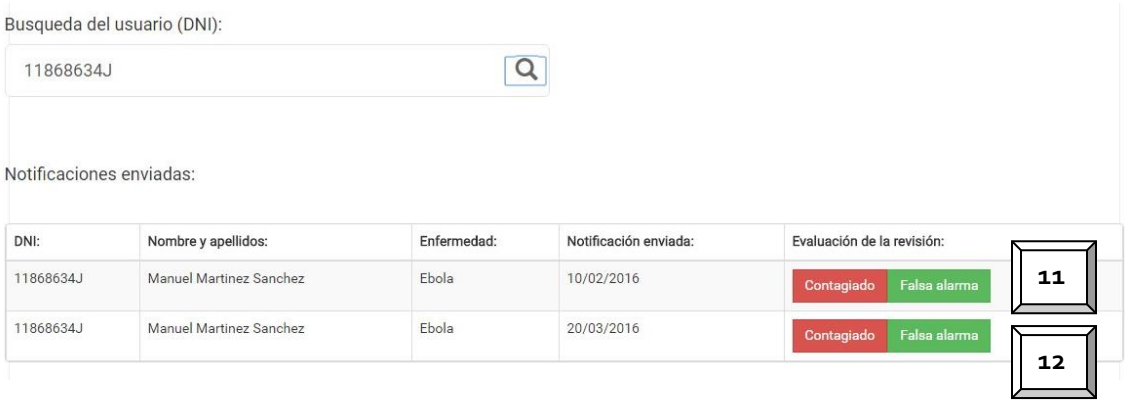


Figura B.2.6 - Manual: Confirmar/Falsa alarma

11 – Una vez buscado el paciente, para confirmar el contagio del paciente pulsar en el botón Contagiado

12 – Para borrar la notificación y confirmar que el paciente está sano y no ha sido infectado, pulsar sobre el botón Falsa alarma

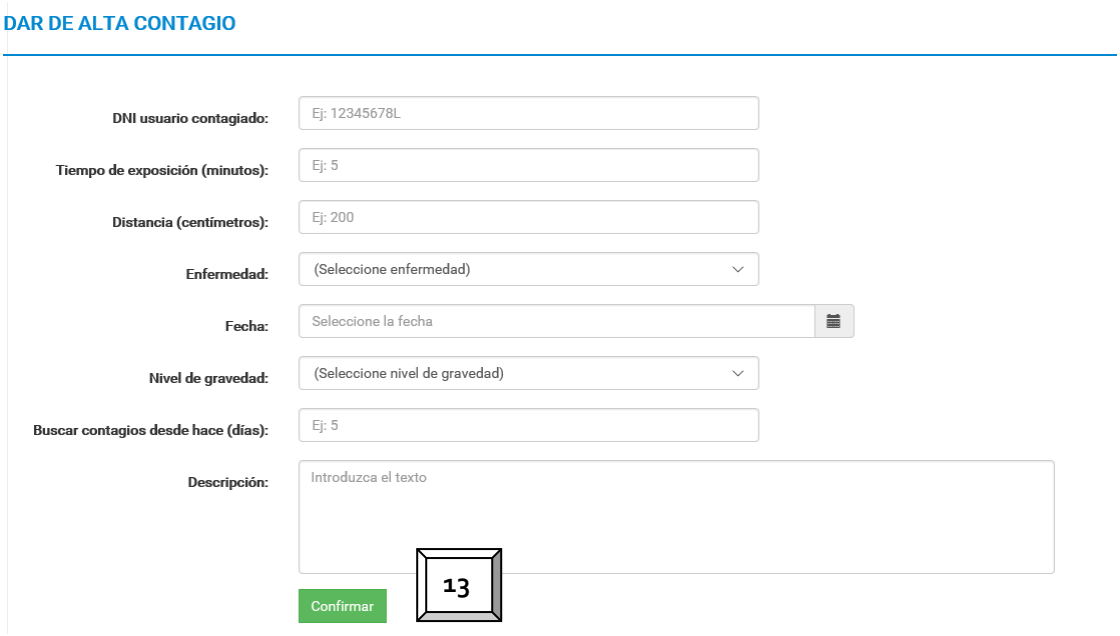


Figura B.2.7 - Manual: Alta contagio

13 – Si se desea dar de alta un contagio, introducir cada campo correctamente (en caso de haber algún error se mostrará un mensaje de alerta) y pulsar el botón Confirmar

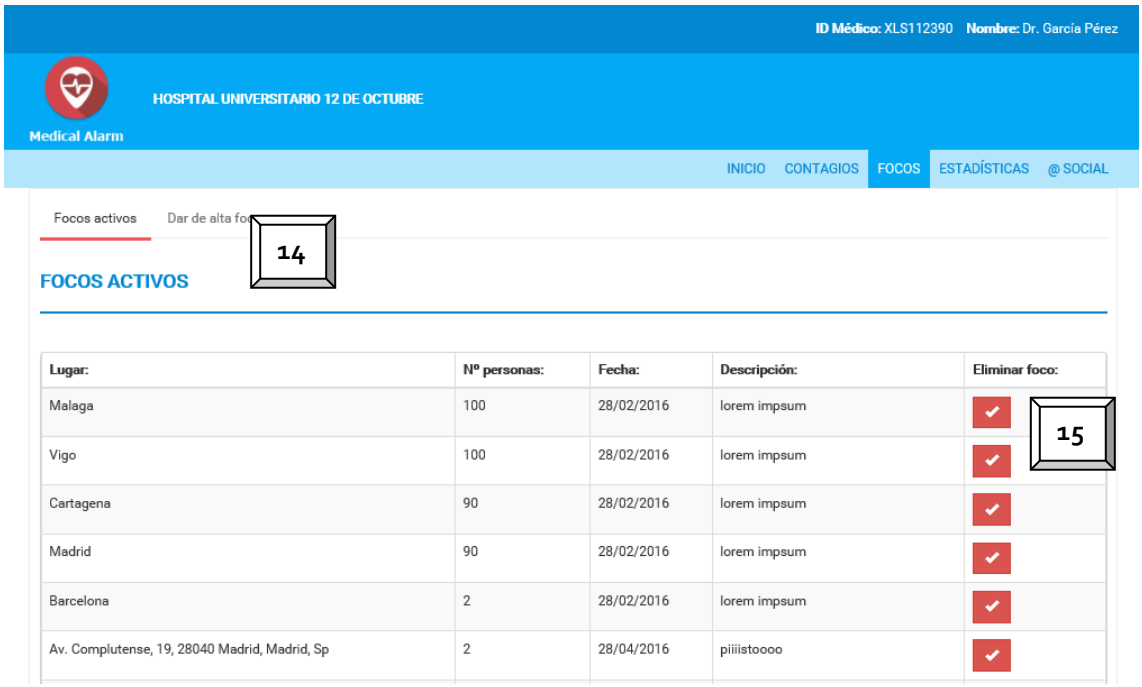


Figura B.2.8 - Manual: Pestaña focos

14 – Dentro de la pestaña de focos, pinchar sobre la sub pestaña dependiendo de la acción a realizar; ver los focos activos en el sistema o dar de alta un nuevo foco

15 – Al abrir la pestaña y cargar automáticamente los focos activos, pulsar en el icono rojo del tick si se desea eliminar el foco del sistema

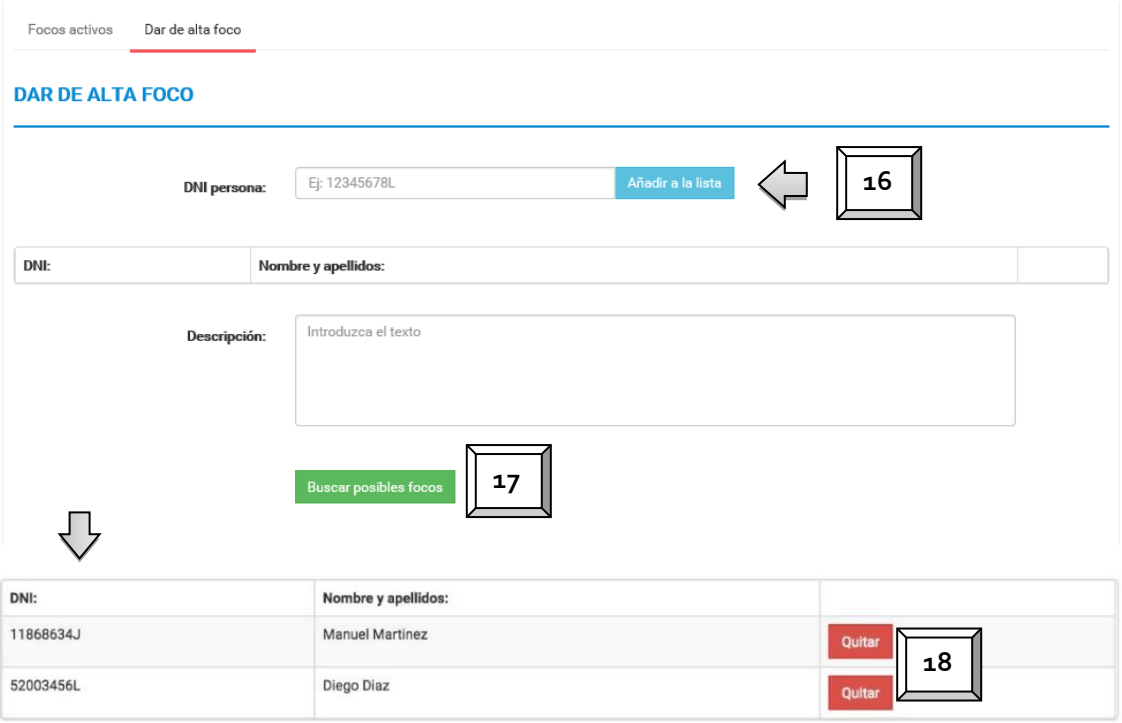


Figura B.2.9 - Manual: Alta foco

16 – Para dar de alta un foco, es decir encontrar la localización exacta donde se ha podido originar un contagio, es necesario al menos dos usuarios. Para incorporar usuarios a la lista antes de lanzar la búsqueda, existe un buscador por DNI. Se debe introducir el DNI completo y sin espacios y pulsar sobre el botón Añadir a la lista. Instantáneamente aparecerá el usuario encontrado en la tabla

NOTA: en caso de introducir un DNI incorrecto o de no existir se notificará mediante un mensaje de aviso

17 – Una vez introducido como mínimo dos pacientes y una descripción, pulsar el botón verde Buscar posibles focos, en caso de querer iniciar la búsqueda

NOTA: si no se han introducido dos usuarios, se notificará mediante un mensaje emergente y si la búsqueda no ha encontrado ningún lugar se avisará de la misma manera

18 – En caso de haber introducido un DNI no deseado o haberse producido cualquier confusión añadiendo a la tabla un usuario no deseado, pulsar el botón Quitar para deshacer la acción y eliminarlo de la misma

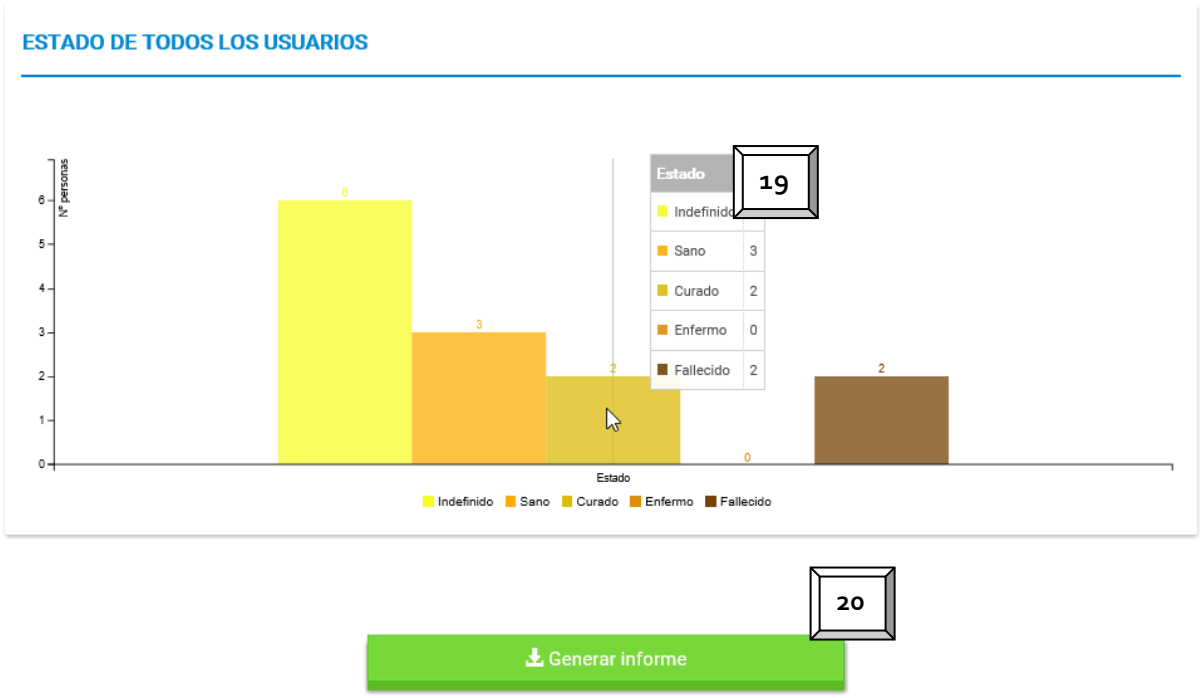


Figura B.2.10 - Manual: Estadísticas usuarios

19 – Pasar el cursor del ratón por encima de las gráficas para ver información detallada de las mismas

20 – Pulsar sobre el botón Generar informe para exportar todos los datos y las gráficas de la página actual en la que se encuentra. Se abrirá una nueva ventana donde podrá seleccionar el formato de exportación y llevar a cabo la acción-

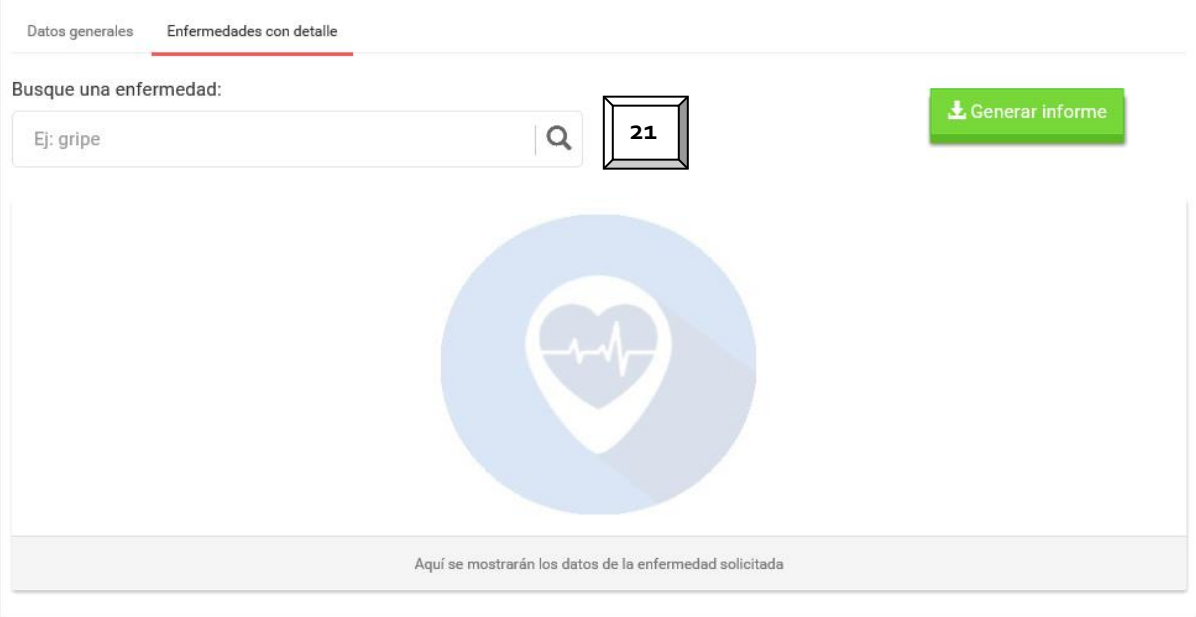


Figura B.2.11 - Manual: Estadísticas de enfermedades en detalle

21 – Dentro de la pestaña de Enfermedades con detalle, si queremos visualizar todos los datos recogidos sobre un enfermedad en particular, existe un buscador en el que introduciendo el nombre de la enfermedad, se muestra toda su información

NOTA: no se aceptan tildes y se comprueba el formato mostrando un aviso en caso de error

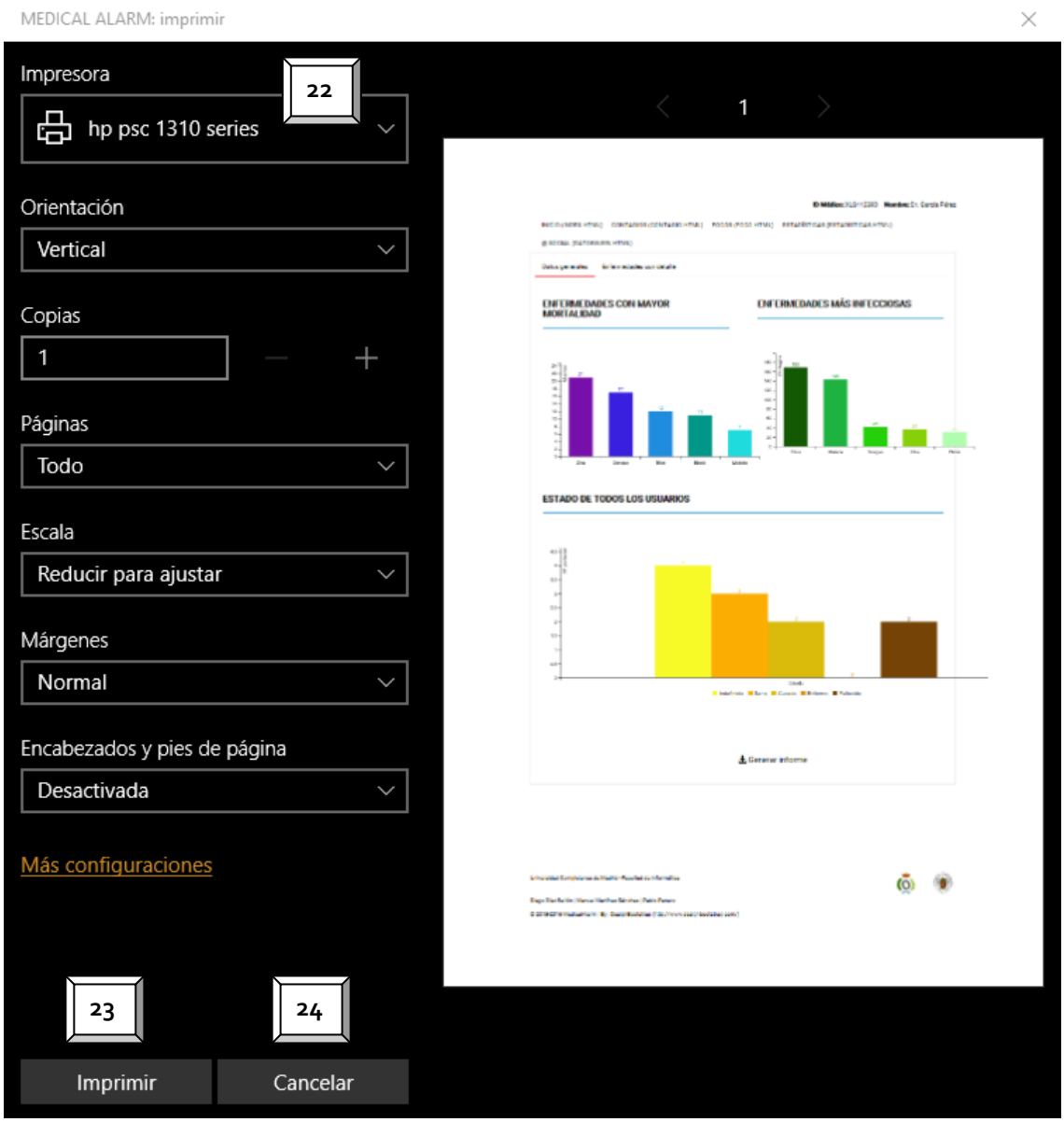


Figura B.2.12 - Manual: Generar informe

22 – Desplegar el menú y seleccionar el tipo de formato que se desea para la exportación. Recomendable el formato PDF universal y portable para todos los sistemas. Completar el resto de opciones disponibles. Para el método estándar dejar las opciones por defecto

23 – Pulsar el botón de Imprimir para iniciar la exportación del documento

24 – Pulsar el botón Cancelar para detener la exportación, cerrar la ventana y volver a la vista anterior

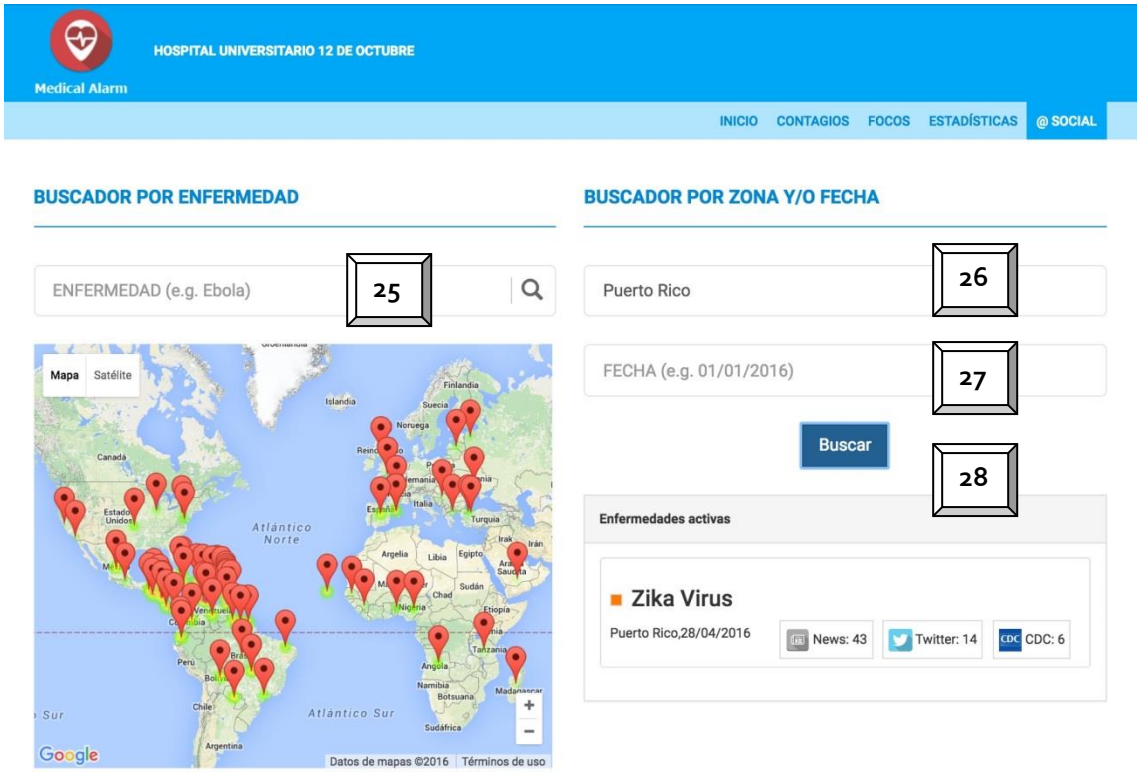


Figura B.2.13 - Manual: Pestaña @social

25 – Para filtrar una enfermedad en el mapa mundial, existe un buscador que permite mostrar únicamente los puntos en el mapa que están contagiados por esa enfermedad facilitando así su visualización y análisis. Introducir el nombre de la enfermedad y pulsar sobre el icono de la lupa para llevar a cabo la búsqueda

NOTA: el buscador no permite tildes ni símbolos extraños mostrando un mensaje en caso de producirse un error. Si en el buscador no se introduce ningún nombre y se pulsa la lupa para buscar se muestran todas las enfermedades y todas las localizaciones existentes en el sistema sin realizar ningún filtro. En ningún caso se muestra un error o mensaje de alerta

26 – En caso de querer filtrar por una zona en particular, introducir en el primer recuadro del panel derecho el nombre de una zona concreta (ciudad, municipio, barrio…)

NOTA: no se aceptan tildes, por lo que se deberán obviar en el buscador. Sí se aceptan mayúsculas y espacios para los nombres propios e iniciales de palabra

27 – En caso de querer filtrar a partir de una fecha en particular, introducir en el segundo recuadro del panel derecho una fecha. Se mostrarán sólo las enfermedades en las que se han producido contagios a partir de la fecha introducida y no anterior a ella

NOTA: prestar atención al formato de la fecha. Sólo se acepta introducir fechas separadas por barras ("slash") y cualquier otro formato como guiones o puntos será inválido

28 – Al pulsar el botón Buscar se muestra la lista de enfermedades y toda la información recopilada de los medios sociales correspondientes que cumplen con los requisitos introducidos en los filtros superiores (zona y fecha). Puede dejarse uno vacío, en tal caso se ignorará y sólo se filtrará por el que contenga un valor válido

Anexo

Se muestra y se adjunta el justificante correspondiente a la reunión con el médico Francisco López Medrano en el Hospital Universitario 12 de Octubre de Madrid con el objetivo de compartir el proyecto con un profesional de la medicina y obtener una opinión técnica y práctica del mismo.

Hospital Universitario 12 de Octubre
SaludMadrid

Comunidad de Madrid

Dr. Francisco López Medrano
Unidad de Enfermedades Infecciosas

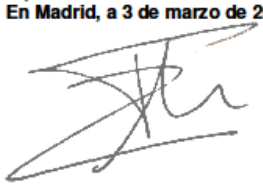
Madrid, 3 de Marzo de 2016

Diego Díaz Bailón, Manuel Martínez Sánchez y Pablo Panero, estudiantes del Grado en Ingeniería Informática de la Universidad Complutense de Madrid, e integrantes del grupo del Trabajo Fin de Grado "Sistema de Control de Enfermedades Contagiosas" solicitaron el asesoramiento técnico-científico del Doctor Francisco López Medrano como especialista en Enfermedades Infecciosas.

El 1 de Marzo de 2016, Manuel Martínez Sánchez, en representación del grupo, acudió a la Unidad de Enfermedades Infecciosas del Hospital Universitario 12 de Octubre de Madrid con el objeto de recibir dicho asesoramiento.

Previa explicación del proyecto, el Doctor López Medrano expuso, como experto, su perspectiva, detallando aspectos legales y posibles aplicaciones prácticas del trabajo.

A petición de los interesados, se emite el presente justificante.
En Madrid, a 3 de marzo de 2016



Fdo. Francisco López Medrano

Médico Adjunto. Unidad de Enf. Infecciosas. Hospital Universitario 12 de Octubre, Madrid
Profesor Asociado. Departamento de Medicina. Facultad de Medicina.
Universidad Complutense de Madrid

Avenida de Córdoba, s/n.
28041 MADRID
Tel.: 91 390 80 00
Fax: 91 469 57 75